# ABSTRACT

ZHU, MU. Defensive Deception in Enterprise Networks: Adaptive Honeyfiles, Decoy Network Flows, and Honeypots. (Under the direction of Munindar P. Singh.)

Defensive deception offers a unique approach to cybersecurity. Unlike traditional security measures, such as firewalls and intrusion detection systems (IDS), defensive deception involves the defender engaging in the attack process to deceive and misdirect attackers. This makes it particularly effective in detecting and mitigating stealthy attacks, such as malicious reconnaissance and insider threats. Two main promising directions to develop defensive deception techniques are observed in the literature. First, the strategies of an attacker and defender are commonly modeled using game theory, where the defender adopts deception strategies with the aim of confusing or misleading attackers into choosing suboptimal or poor strategies. Second, deception techniques based on machine learning create fake information or decoy objects that mimic real objects to mislead or lure attackers. To better understand defensive deception, we survey existing research to identify the uses of game theory and machine learning, the types of attacks discussed, the deceptive techniques proposed, and methods of evaluation.

Honeyfiles are a popular defensive technique to detect threats such as insider attacks. However, honeyfiles can disrupt legitimate uses and overload administrators by generating false alarms. To address this problem, we develop a novel honeyfile system called Mee. Mee is based on decentralized deployment and centralized control. We also develop a Bayesian game to model interactions between attackers, defenders, and users to help the defender select an optimal strategy. Empirical results through simulation show that Mee performs better than traditional honeyfile systems and causes less disruption for legitimate users. The Mee approach is limited in that its game theory model can only be applied to relatively simple scenarios, such as those involving a few players. Therefore, we propose an enhanced approach called HoneyMee that uses Deep Reinforcement Learning (DRL) and a more complex environment involving multiple users and attackers simultaneously. We construct a DRL model for interaction between legitimate users, insider attackers, and defenders. Empirical results through simulation show that HoneyMee greatly improves the accuracy of insider threat detection and the payoff for the defender.

Another contribution investigates a hybrid honeypot system that combines low and high-interaction honeypots to deceive attackers. We propose a two-player hypergame model to determine how the defender should deploy the honeypots to protect the network from malicious reconnaissance. Our model considers the imperfect knowledge of each player about their opponent and examines the payoff between them. We also analyze the best strategies for each player within the hypergame framework.

An inside attacker can conduct malicious reconnaissance of network traffic. We investigate an approach to generate honey traffic, i.e., fake network flows, to defend against such an attacker. This research adopts the Generative Adversarial Network (GAN) architecture, which includes a generator and a discriminator component. We use the generator to learn the features of network flows that come from actual servers and produce fake flows. In addition, we use the discriminator to mimic an attacker, which needs to distinguish legitimate flows from fake flows.

# Defensive Deception in Enterprise Networks: Adaptive Honeyfiles, Decoy Network Flows, and Honeypots

by
Mu Zhu

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2024

APPROVED BY:

| | |
|---|---|
| Jin-Hee Cho | William Enck |
| Rudra Dutta | Khaled Harfoush |

Munindar P. Singh
Chair of Advisory Committee

## BIOGRAPHY

Mu Zhu was born in the Chinese city of Luoyang, which was a prosperous capital in multiple dynasties. He was raised in Zhengzhou and finished high school there. He then obtained a dual Bachelor's Degree in Electronic Engineering and Finance from Zhengzhou University.

With a wish to explore a "bigger world," Mu moved to the United States and earned a Master's Degree in Computer Engineering from the University of Delaware. Fascinated by the topics of cybersecurity and encryption, he devoted much of his time during his Master's studies to learning about them. During his doctoral studies, Mu is being mentored by Dr. Munindar P. Singh and is researching cybersecurity. Sun Tzu's famous book, *The Art of War*, states that all warfare is based on deception. Mu accepts this notion, and thus his primary area of research is defensive deception, which is used to disorient, entice, and deceive attackers in the digital realm.

## ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my family for their constant support throughout this endeavor. Their love, motivation, and empathy have been my sources of strength, and I am truly thankful for their numerous sacrifices. I am particularly grateful for the understanding and support of my wife, Tingting Zhu, without whom I could not have completed my studies or established our happy home.

My heartfelt gratitude goes to my advisor, Dr. Munindar P. Singh, for his invaluable assistance, direction, and patience during my research. Without his guidance, this work would not have been possible.

I also want to thank all the collaborators and peers who contributed to this research. I gained valuable insights from Dr. Jin-Hee Cho, who provided guidance on game theory, machine learning, and defensive deception methods. I am also thankful to Dr. Charles Kamhoua, Zelin Wan, and Ahmed Hemida for their collaboration, support, and shared knowledge that enhanced the quality of my study and research.

I appreciate the members of my dissertation committee, Drs. Jin-Hee Cho, William Enck, Rudra Dutta, Khaled Abdel Harfoush, and Munindar P. Singh, for their time, expertise, and constructive feedback. Their guidance and recommendations were crucial in refining this dissertation.

The Army Research Office supported this work under awards W911NF-17-1-0370 and W911NF-20-2-0140 . The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes without standing any copyright notation.

This achievement would not have been possible without the backing and motivation of all those mentioned above. Thank you for believing in me and being part of this significant milestone in my academic journey.

## TABLE OF CONTENTS

## LIST OF FIGURES

**LIST OF TABLES**

# CHAPTER 1 INTRODUCTION

The use of defensive deception to address cybersecurity has become increasingly popular. This approach involves deceiving attackers by luring, misleading, or confusing them. As a result, defensive deception provides a unique way to detect and counter stealthy threats such as advanced persistent threats (APTs) and other insider attacks. This chapter covers the strategies of deceit and models of threats that are the main focus, along with the rationale behind our research and distributions. We explain why we view the insider attacker as the threat model and why we employ game theory and machine learning to help the defender choose the best solution.

## 1.1 Defensive Deception

Deception has been applied broadly and with a variety of meanings. Deception is a tactic that has been employed in many different fields, such as the military, criminology, and economics. It involves one party providing false information to gain an advantage over the other. This strategy is used to create confusion and gain an upper hand in a situation.

In cyberspace, both attackers and defenders can use deception to fool their opponents. Malicious deception tactics, including phishing emails, false identities, and man-in-the-middle attacks, are often employed by attackers to deceive defensive systems. For example, an attacker may send phishing links to acquire valid credentials from victims or pretend to be a legitimate user to gain access to a system. In particular, malicious deception is frequently seen in the reconnaissance and exploitation stages of prolonged attacks, such as Advanced Persistent Threats (APTs).

Our study focuses on cyberdeception, which involves using misleading information to protect against attackers. Defensive deception involves the defender altering the appearance of relevant assets to conceal valuable information or expose the movements of the attacker. Although the attacker may be able to use strategies to identify the deception, it slows down their attack and increases the cost for them.

Furthermore, the defender can use mock objects to attract the attention of attackers. Popular luring-based deception methods include honeypots and honeyfiles, aimed at wasting the attackers' time and detecting their behavior. These deception techniques can also be used to monitor the attackers and learn about their interests. The main goal of defensive deception is to intentionally create false information, leading the attacker to acquire or maintain a false belief rather than the actual perspective of the system. Defensive deception has a wide range of applications in the field of cybersecurity. It can be used to proactively protect systems, prevent malicious activity, deter attackers, deflect attacks, and detect malicious activity.

- Proactive Protection: Proactive defense involves taking steps ahead of time to anticipate and counteract any potential threats before they become a reality [Colbaugh and Glass, 2011]. Defensive deception includes a proactive approach to introducing uncertainty and false information into the environment. This is done by adding deceptive elements, such as honeypots and fake network flow, to create a more complex and unpredictable environment for attackers. The aim is to disrupt their reconnaissance activities and cause confusion. Additionally, honeypots can be beneficial to the defender, as they can be used to observe the attacker and gain knowledge about their techniques, such as the tools they use and their objectives, and to identify the system's weaknesses and valuable assets. In Chapter 4, we present a hypergame-based hybrid honeypot system to counter passive monitoring and active probing by attackers. In Chapter 5, we explore the use of GanFlow to confuse an attacker's reconnaissance activities.

- Prevention Defense: The prevention defense strategy focuses on reducing the chances of successful cyberattacks. To prevent successful attacks, deception tactics can be used, such as leading attackers away from genuine assets, providing them with false information, or using honeypots to keep attackers away from real assets. This helps protect real assets and provides an extra layer of security. In Chapter 3, we present our research on honeyfiles, which can be used by defenders to detect malicious insiders.

- Deflection: This approach involves leading attackers away from essential resources or vulnerable areas. Deception can be employed to draw attackers away from vital assets or weak spots. False objectives and misdirection strategies can be used to guide attackers towards dummy systems, lessening the effect of the attack and giving security teams more time to react.

- Detection: The identification and response to security incidents are improved by deception. Triggers or alerts are created when attackers interact with deceptive elements, such as honeypots and honeytokens, which can alert of potentially malicious activity and enable a quick response before any real assets are damaged. Honeyfiles are also useful for detecting insider threats.

- Deterrence: The aim of deterrence is to dissuade potential aggressors by increasing their perceived risks and uncertainties. Cyberdeception can be employed to accomplish this by making it more difficult for attackers to anticipate the success of their actions, thus discouraging them from targeting a certain system or network. Additionally, deploying cyberdeception can raise the cost of attackers when they intend to execute malicious activities. For example, attackers must invest more time and effort to conduct reconnaissance to distinguish between fake and real hosts, which also deters attackers.

In various application contexts, defensive deception is frequently utilized in network security setups to identify and counteract cyber threats. This strategy involves strategically placing deceptive elements like honey files, honeypots, and honey flows throughout the network infrastructure. In endpoint security, deception techniques, such as honeyfiles and fake account numbers or passwords, can be implemented at the individual device level to detect and respond to malicious activities. Moreover, as cloud computing becomes more prevalent, defensive deception methods can be expanded to cloud environments to improve security measures and identify unauthorized access attempts. In a tactical setting, defensive deception proves to be more effective than other defensive strategies. Deception technologies offer the advantage of not only detecting or preventing attacks but also misleading and confusing attackers. This allows the defender to continuously monitor attackers, understand their techniques and interests, and waste attackers' efforts.

Defensive deception techniques are commonly used to counter various cyber threats, including:

- External Attacks: Such as targeted malware infections, phishing schemes, and network intrusions initiated by external threat actors.
- Insider Threats: Unauthorized access attempts by employees, contractors, or partners with malicious intent or compromised credentials.
- Advanced Persistent Threats (APTs): Complex, long-term cyber espionage campaigns carried out by well-resourced adversaries aiming to infiltrate and compromise specific organizations.
- Bot Attacks: Including distributed denial-of-service (DDoS) attacks, credential stuffing, and brute-force attacks launched by automated bots exploiting vulnerabilities in network defenses.

### 1.1.1 Honeyfile

Honeyfiles, also known as decoy documents, are fake files that are used to deceive attackers [Yuill et al., 2004]. Compared to other deception techniques, such as honeypots, the benefits of honeyfiles are that they are lightweight and easy to deploy and maintain. Research into honeyfiles has two main approaches: 1) the generation of honeyfiles [Abay et al., 2019; Whitham, 2016, 2017] and 2) the placement and quantity of honeyfiles [Bowen et al., 2009; Gómez-Hernández et al., 2018; Voris et al., 2015; Zhu and Singh, 2022]. The first approach looks at how the system can generate suitable honeyfile content using natural language processing. The second approach examines the appropriate number of honeyfiles and other characteristics, such as placement and name, to make them more attractive.

A honeyfile alarm may indicate the presence of an attacker, but it could also be a false alert, meaning that a user unintentionally interacts with it. The defender must have the ability to distinguish between alarms in order to reduce the rate of incorrect decisions. On the other hand, the location and number of honeyfiles can affect the efficiency of honeyfiles. For example, a large number of honeyfiles may raise the likelihood of detecting an insider attacker, but also increase the cost of upkeep and lead to confusion for legitimate users.

**Limitations:** While honeyfiles can be efficient in identifying unauthorized access or malicious behavior within a network by serving as decoy files, they come with limitations. Initially, honeyfiles may not always replicate real files accurately, which could result in false positives or negatives. Designing convincing honeyfiles that appear authentic to potential intruders while remaining distinguishable to security personnel can be a complex task. Moreover, the efficacy of honeyfiles is highly dependent on the sophistication of the attackers. Proficient adversaries may be able to identify and avoid honeyfiles, reducing their effectiveness as a detection method. Additionally, the deployment and maintenance of honeyfiles require meticulous planning and continuous maintenance. Without proper care, they may become outdated or ineffective over time. Finally, while honeyfiles can offer valuable insights into the strategies and methods employed by attackers, they should be integrated into a comprehensive defense strategy that encompasses other security measures like firewalls, intrusion detection systems, and staff training. Recognizing these constraints ensures a thorough comprehension of the role honeyfiles play in a cybersecurity strategy and assists organizations in making well-informed decisions regarding their utilization.

### 1.1.2 Honeypot

Honeypots are a widely used defensive deception technique that can be used to attract, deceive, and detect threats. When deployed correctly, honeypots can divert attackers' attention away from valuable assets and provide early warnings about new attacks and exploitation. By using honeypots, defenders can uncover malicious activities and gain insight into attackers' behavior during and after exploiting a honeypot. Honeypots can be classified according to the level of interaction.

**Low-interaction honeypots** simulate a few internet protocols and network services. They can imitate protocols such as TCP and IP, deceiving attackers into thinking they are connecting to a genuine environment instead of a honeypot. Although low-interaction honeypots are simple to install and manage, they offer limited capabilities and are not effective against experienced attackers or sophisticated threats such as zero-day attacks.

**High-interaction honeypots** attract attackers by providing them with (realistic) vulnerable services or software. These honeypots are more sophisticated than low-level honeypots and can imitate existing systems to a greater extent. They are beneficial in detecting zero-day attacks and recognizing unknown vulnerabilities. However, they require greater resources to maintain than low-level honeypots.

Researchers have sought to combine the characteristics of high and low-interaction honeypots to create a hybrid honeypot system, or honeynet [Artail et al., 2006; Wang and Wu, 2019]. This approach involves deploying low-interaction honeypots that emulate services and operating systems and then directing malicious traffic to high-interaction honeypots. Here, attackers interact with real services, allowing for the recording and analysis of their activities. This information can then be used to take protective measures for the network.

**Limitations:** Honeypots play a crucial role in cybersecurity, yet they come with limitations, especially when implemented in real-world scenarios. Managing and monitoring honeypots can be resource-intensive, requiring significant time, commitment, and expertise, which can be a challenge in environments with limited resources. Honeypots also have the potential for generating false positives or false negatives, leading to wasted time investigating non-existent threats or overlooking genuine ones. Skilled attackers may be able to detect and bypass honeypots, especially if they are not properly configured or regularly updated. Relying solely on honeypot data may create a false sense of security. Deploying honeypots in certain jurisdictions may raise legal and ethical concerns, as they could attract malicious actors and inadvertently facilitate attacks on other systems, necessitating compliance with relevant laws and regulations. Honeypots provide insights only into the specific area they are monitoring and may not detect threats targeting other network segments or employing different attack tactics. Maintaining effective honeypots requires regular updates and maintenance, including updating signatures, managing decoy services, and analyzing captured data, which can be demanding in practical settings where IT teams are often under pressure. Establishing and sustaining efficient honeypots can be costly, particularly for organizations with limited budgets, encompassing not only initial setup expenses but also ongoing costs for maintenance and oversight. Despite these challenges, honeypots can still be valuable tools when integrated into a comprehensive cybersecurity strategy. Understanding their strengths and weaknesses, and combining them with other security measures, enhances the overall defense strategy.

### 1.1.3 Decoy Network Flows

Decoy network flows, also referred to as honey flows, replicate or mimic synthetic routes or data streams across the network framework [Anjum et al., 2021; Miah et al., 2022]. These flows are designed to imitate genuine traffic behaviors, applications, or services, diverting potential attackers towards decoy endpoints or honeypots. Acting as a diversion strategy, decoy network flows lure adversaries into interacting with misleading components instead of the real network infrastructure. Through monitoring and scrutinizing the activities within these decoy flows, security teams can acquire valuable information on the strategies and methods utilized by adversaries, thereby improving threat detection and response abilities.

**Limitations:** Generating and maintaining realistic decoy network flows can require a considerable amount of computational and network resources. It can be a challenging task to produce sufficient traffic that mirrors genuine activity without causing performance issues or network congestion. Additionally, the creation and supervision of decoy network flows require a deep understanding of network protocols, behaviors, and traffic trends. Establishing convincing decoys that simulate legitimate services and applications can be intricate and time-consuming, especially in dynamic network environments where traffic patterns are in constant flux. Moreover, these decoy network flows may not offer a comprehensive insight into all aspects of attacker conduct. Intruders might bypass or escape the decoys, particularly if they identify irregularities or anomalies that hint at the presence of deceit. Additionally, adept attackers could engage in reconnaissance to pinpoint and steer clear of decoy endpoints, thus reducing the effectiveness of this deception strategy.

Although honeyfiles, decoy network flows, and honeypots share the common objective of defensive deception, they function at varying levels and play complementary roles in the network defense strategy. Honeyfiles and decoy network flows are mainly active at the data and network layers, enticing attackers and redirecting their focus from vital assets. On the other hand, honeypots operate at the system or application layer, offering a more immersive setting to interact with adversaries and collect in-depth information on their strategies and goals.

In summary, Honeyfiles, decoy network flows, and honeypots are essential elements of a robust defensive deception plan in networks. Through the strategic placement and incorporation of these components, defenders can efficiently identify, discourage, and react to malicious individuals, ultimately enhancing their cybersecurity measures and protecting crucial resources from emerging risks.

### 1.1.4 Advantage and Limitation of Defensive Deception

In this section, we explore the advantages and disadvantages of creating and using defensive deception.

**Key Advantages of Defensive Deception Techniques:**

- Defensive deception techniques provide a high degree of accuracy in recognizing malicious activities. For example, firewalls or intrusion detection systems (IDSs) may create multiple false alarms, but alerts from honeypots are more likely to detect potential threats.
- The use of defensive deception is an effective way to protect against stealth attacks. Traditional cybersecurity approaches are not sufficient to detect hidden risks, such as malicious reconnaissance and insider threats. However, deception techniques (e.g., honeyfiles and honeypots) provide a fresh outlook for observing the behavior of attackers or provide false information to mislead attackers' beliefs.
- Defensive deception can be used to supplement traditional security. For example, honeypots can be used to incorporate Intrusion Detection Systems (IDS) and other monitoring tools to detect intrusions and gain insight into attack characteristics, thus protecting the system from malicious attacks before they reach their intended targets.
- Defensive deception can be used in a variety of ways across multiple layers of systems, including networks, applications, and data. The ease of deployment of these methods also enhances flexibility and adaptability when it comes to providing multiple levels of protection.
- Defensive deception is cost-efficient in defending against cyberattacks when compared to other strategies. Its implementation cost is relatively low, yet it is highly effective in deceiving attackers. For example, honey files or honey tokens are simpler to deploy and maintain than other traditional cybersecurity measures, such as access control or intrusion detection.
- Defensive deception provides greater information about an attacker's activities and assumptions than conventional defense mechanisms, such as intrusion detection or prevention. For instance, honeypots are a widely-known and effective monitoring tool that can give additional attack characteristics to help the defender gain a better understanding of the threats, including the attacker's actual objectives and tools, rather than just blocking them.
- The implementation of automated cyberdeception strategies, such as concealing session information, changing data routes, and making software code hard to interpret, has significantly improved security protocols to prevent automated attacks.

**Key Limitations of Defensive Deception Techniques**

Although defensive deception can be used to confuse attackers and slow down malicious processes, it has certain drawbacks that motivate us to enhance deception techniques.

- It is difficult to determine the purpose, aim, and objective of an attacker when they launch and carry out an attack, making it hard to select the most suitable defensive deception strategy based on the estimated risk and benefit of the strategy.

- Most honey-X techniques (e.g., honeypots, honeyfiles, honey tokens) are designed to deceive attackers by providing them with false information, which may lead them to make suboptimal or poor choices when launching attacks. To prevent normal users or defenders from being misled, extra procedures or protocols may need to be implemented.
- The need for continual adjustment of defensive deception is a result of its deceptive nature. Without regular reconfiguration and implementation, attackers would be able to identify deceptive devices with ease, unless the deception is automated, similar to software obfuscation techniques.
- As attackers become more sophisticated, there is an increased need for complex and high-quality deception tactics that are difficult to detect. This makes defending a system more expensive and more challenging to manage than a traditional system without any deception defense mechanisms.
- Some defensive deception techniques can also be disruptive to legitimate users or administrators. For example, legitimate users may inadvertently interact with honeyfiles, resulting in false positive alerts, which can be disruptive to users and can lead to confusion for the defender in accurately assessing the system's security status.
- Given the nature of deception, the effectiveness of such techniques requires a continual process of configuration, reconfiguration, and implementation. Otherwise, attackers can easily identify deceptive devices, unless the deception is automated, such as through software obfuscation. This can be costly; for instance, a high-interaction honeypot can provide valuable information about an attacker when it is successful in luring them, but it requires resources to maintain, thus increasing the cost of a security system when there is no attacker.
- With well-trained attackers, there are growing demands for elaborate deception strategies (e.g., high-interactive honeypots) that attackers cannot easily identify. However, doing so brings a higher defense cost and greater difficulty in managing a defense system than a traditional system without deception mechanisms.

The need to improve defensive deception techniques is highlighted by these limitations. For example, how can the defender deploy honeypots to increase the chances of detecting an attacker while minimizing the cost of maintaining them? Additionally, how can the defender determine the number of honeyfiles that will be effective in confusing an insider attacker while not having too much of an impact on legitimate users?

## 1.2 Threat Models

This section presents the primary threats on which our research is focused.

### 1.2.1 Advanced Persistent Threat (APT)

Advanced Persistent Threats (APTs) are a growing concern, as they are highly skilled attackers who can carry out long-term attacks to gain access to valuable and confidential economic, proprietary, or national security information. APTs are considered a major security risk, and their numbers are increasing rapidly, posing a threat to networks and systems around the world [Bandla, 2019]. As a result, APTs are receiving increasing attention from researchers. To describe an APT, researchers [Chen et al., 2014] define six stages of an APT's cyber kill chain:

- Reconnaissance: At this stage, attackers collect data, such as email addresses and social ties of users. With these data, the attacker can pinpoint potential targets within the targeted organization.
- Delivery: At this stage, attackers launch their exploits against targets. Two common approaches are *spear phishing*, which involves sending emails with malicious attachments or links, and *watering hole attacks*, which involve using third-party applications or websites that the victim visits regularly to deliver malware. APT attackers generally use the data collected during reconnaissance to make the attack more tailored and personal to the victim.
- Initial intrusion: Attackers who use APT techniques breach the target device and carry out exploits. In some cases, they may gain authorization through earlier stages and access the device as a legitimate user. This stage involves the attacker setting up footholds, such as backdoors, in the target network.
- Command and control (C2): Attackers use controlled servers to send commands to compromised systems and receive data from a target network through C2 mechanisms.
- Lateral movement: Once the connection between the target network and the C2 server has been established, the attacker can begin to infiltrate the network and extend their control across the organization. This phase typically involves a few goals: conducting internal reconnaissance, gathering additional credentials, compromising more systems, and locating valuable data or assets.
- Data exfiltration: This step involves gathering, encoding, and taking out pilfered information from the target's surroundings.

### 1.2.2 Insider Attack

Salem et al. [2008] divide insider threats into two types: *traitors* and *masqueraders*.

**Traitors** are individuals who are legally authorized to access an organization's data but use that access to collect and distribute sensitive information. Additionally, they use their legitimate status to carry out malicious activities, such as unauthorized access and alteration of confidential information, which can result in loss of integrity and availability of data.

**Masqueraders** are attackers who acquire legitimate users' credentials to gain access to a target environment and steal valuable data from the victims' devices. Advanced Persistent Threat (APT) attackers can become masqueraders by collecting enough authorizations or establishing backdoors in the victims' devices.

These two types of insider threats have different perspectives about the victims. Masqueraders may be able to gain access to devices, but they are not familiar with the victim system, such as the file structure. This provides the defender with the chance to detect the intruder's activities. For instance, a masquerader is more likely to interact with honeyfiles than a legitimate user as it needs more time and steps to explore the victim system.

In Chapter 3, we address masquerader-type insider attackers as the threat model. We assume that these attackers have the capability to infiltrate devices and look for files that are of interest to them. We also assume that an insider attacker requires more time and effort than a legitimate user to investigate the file system, which may increase the likelihood of the attacker coming into contact with a honeyfile.

### 1.2.3 Malicious Reconnaissance

Reconnaissance in the context of cybersecurity is a critical element, whether it is active or passive and legal or malicious.

**Passive monitoring** involves monitoring or scanning, which entails observing the traffic generated by servers and clients as it passes an observation point to identify active services [Schuster et al., 2017]. This type of reconnaissance is difficult to detect and requires fewer network resources than active probing, making it suitable for long-term use as part of regular operations. Passive scanning can detect active nodes, their services, supported protocols, operating systems (OSs), enterprise roles, and update schedules [Montigny-Leboeuf and Massicotte, 2004]. Passive monitoring is nonintrusive and generally invisible to the hosts running the services, making it difficult to detect by traditional security measures. Common passive monitoring techniques include button grabbing and packet sniffing, which focus on searching services by analyzing packet length, timing, web flow size, and response delay. However, passive scanning can detect only active services and those running on well-known ports, and using protocol-specific decoders. Firewall configurations can catch services that an active probe attack misses.

- Banner grabbing can be useful to both network administrators and remote attackers. Through this technique, attackers can uncover sensitive information about the services running on a system.
- Packet sniffing is the technique of intercepting network traffic by capturing and storing data as it is transmitted. This method enables the reconstruction of network packets and the viewing of the information that was sent over the network. It is important to be aware that plain text data, such as usernames, passwords, and IP addresses, can be intercepted and accessed by unauthorized individuals. Therefore, it is essential to take steps to safeguard sensitive information from being eavesdropped.

Montigny-Leboeuf and Massicotte [2004] investigate the data that can be acquired through passive monitoring, including active nodes, operating systems, and IP network configuration. Additionally, they discussed the advantages of passive network discovery and persistent monitoring in terms of providing useful contextual information.

In Chapter 5, we analyze the threat of passive monitoring. Attackers can gain access to switches to observe and analyze packets, thereby uncovering network topology and device vulnerabilities. To protect real hosts, the defender can deploy fake flow to disrupt the attackers' observation.

**Active probing** involves sending packets to a host and analyzing the response. This enables an attacker to identify the vulnerabilities and importance of a node [Fu et al., 2004]. Probing can be tailored to a particular protocol or application. It is a fast process and provides a comprehensive report of all open and unprotected ports [Bansal et al., 2012]. However, its intrusive nature can be easily detected. In a cybergame setting, we will consider attackers who use a combination of passive monitoring and active probing [Bansal et al., 2012; Fu et al., 2004] as their attack strategies.

In Chapter 4, we explore how attackers can use active probing and passive monitoring to gather information to identify potential targets to attack. We also examine how defenders can use a hybrid honeypot system to protect against reconnaissance.

## 1.3 Defensive Deception based on Game Theory

Game theory has been an effective mathematical discipline for resolving strategic decisions in a variety of dynamic contexts. It has been used to generate mathematically validated solutions to intricate, dynamic decision-making situations. Nevertheless, one of its common drawbacks is that it depends on stringent assumptions to derive Nash Equilibria, such as shared knowledge about the probability distributions of Nature's moves or players' accurate beliefs towards opponents' moves. Since its origin in the 1700s [Bellhouse, 2007], game theory has advanced significantly, taking into account increasingly realistic facets of real-world problems. Examples of this include the consideration of limited rationality, decision making under uncertainty using the concept of mixed strategies, and games of imperfect or incomplete information.

Bayesian games are a kind of strategic interaction in game theory in which players have limited knowledge of the game's parameters and hold beliefs about the potential states of the world. The decisions of the players are influenced not only by their preferences but also by their beliefs about the probable states of the world and the actions of other players. Furthermore, in a Bayesian game, the players may be assigned different types at the start of each game [Dekel et al., 2004; Harsanyi, 1967]. Because of these features of Bayesian games, in Chapter 3, we model the interactions between users, attackers, and a defender. In our game model, one player represents the defender, while another player may be an attacker or a user. The advantage of using Bayesian games is that we can consider users as one type in the game rather than just a game between defenders and attackers.

Hypergames represent multiple players who have different perspectives, games with partially observable Markov Decision Processes (POMDPs) that involve subjective beliefs and rationalities [Tadelis, 2013]. Due to the characteristics of the hypergame model, it is effective in taking into account defensive deception strategies as a defensive measure, since defensive deception can lead to different interpretations of the environment by attackers and defenders. In Chapter 4, we use hypergame to model the interaction between a malicious reconnaissance attacker and a defender, taking into account the uncertainty of the attacker and the defender's beliefs.

## 1.4 Defensive Deception based on Machine Learning

Game theory offers the defender the chance to explore more effective strategies to improve defensive deception. However, there are restrictions to the use of game theory in defensive deception studies, such as the ability to model only basic scenarios and limited model parameters. Machine learning is becoming common in many different areas, especially in the field of cybersecurity. Machine learning methods have been widely used to automate attacks and to gain insight into system behavior in the cybersecurity domain [Al-Shaer et al., 2019]. Machine learning algorithms can be used to create deceptive patterns and to adaptively respond to changes in attacker behavior. Machine learning models can be used to analyze normal and abnormal behavior within a network, helping detect potential attacks and trigger deceptive responses.

Deep Reinforcement Learning (DRL) is a branch of machine learning that combines reinforcement learning (RL) with deep learning techniques. It involves teaching artificial agents to make decisions in a given environment by learning from their interactions with it. In Chapter 3.5, we use DRL to improve the research conducted in Chapter 3.4. Both of these approaches make use of a similar environment and deceptive structure, however, Chapter 3.5 takes into account a more realistic situation and provides greater details about the defender, attackers, and legitimate users.

A Generative Adversarial Network (GAN) is a type of artificial intelligence algorithm used in unsupervised machine learning. It is composed of two main components: the generator and the discriminator. The generator's purpose is to generate synthetic data samples that resemble the real data from the training set. The discriminator aims to determine if a given data sample is real (from the true data distribution) or fake (generated by the generator). In Chapter 5, we have created a GanFlow generation structure that employs the generator of GAN to learn from the flow patterns of real hosts and direct honeypots to generate GanFlows. Malicious reconnaissance attackers act as the GAN's discriminator, attempting to differentiate between real and fake flows.

Due to the benefits of game theory and machine learning, numerous studies of defensive deception have employed these approaches to assess or enhance the efficacy of cyberdeception techniques. In Chapter 2, we survey and analyze defensive deception research based on game theory and machine learning to gain a better understanding of current studies, including their model settings, testbed settings, and metrics, and suggest potential future research directions for game theory and machine learning-based cyberdeception.

## 1.5 Thesis Statement

Defensive deception can provide a novel approach to defending against certain attacks. However, it can also increase the load on a system's resources and potentially disrupt the system's legitimate users. We attempt to improve defensive deception by exploring the use of game theory and machine learning. By adopting these two approaches, the defender can gain a better understanding of the environment and its attackers. This understanding can enable the defender to more effectively allocate resources and minimize the disruption to the system's regular operations. The results of our research into defensive deception based on game theory and machine learning have led us to the following thesis statement.

*Defensive deception strategies based on machine learning and game theory help a defender dynamically distribute and deploy decoy resources such as honeyfiles, honeypots, and decoy network flows to increase disruption to attackers while minimizing the effect on legitimate users.*

## 1.6 Research Questions

We refine the above thesis statement to identify the following specific research questions. Given the advantages and difficulties associated with defensive deception, it is essential and worthwhile to improve deception techniques. The purpose of this study is to gain an understanding of existing defensive deception and to enhance certain deception techniques. Accordingly, we identify the following research questions:

**Question 1:** *how should the defender improve its effectiveness in enticing the attacker through deception?*

The main objective of defensive deception is to employ false information to divert the assailant's attention from actual resources. Therefore, the first step is to make the deception more attractive. This question is challenging since it is difficult to understand the attacker's underlying motivations.

**Question 2:** *how should the defender effectively allocate resources?*

Although defensive deception is a cost-effective security measure, the redundant implementation increases the defender's expenses and interferes with the system's normal operations. Specifically, different devices may have different values for the defender. For example, a database server containing sensitive information may be more critical than a website server and thus require more resources to protect. This question is challenging as successful deception tactics require the defender to precisely predict the attackers' actions or behavior.

**Question 3:** *how should the defender reduce the impact of deception methods (e.g., confusing legitimate users)?* The use of defensive deception, such as honeyfiles, can be effective in confusing attackers, but it can also be disruptive to legitimate users. Most research on deception considers only the interaction between attackers and defenders. We address this question by taking into account the legitimate users in the scenario and incorporating false positive rates as one of the metrics in our evaluation.

## 1.7 Contributions

We address our research questions through four projects, which we introduce in this section.

### 1.7.1 Research Approaches of Honeyfile System

While insider attackers have been a focal point in several studies, there has been a lack of discussion regarding the deployment of deception as a defensive measure. Our objective is to investigate the application of defensive deception to counter insider attacks. Honeyfiles are simple to deploy and effective in uncovering hidden attacks. Existing research on honeyfiles often neglects their impact on genuine users and strategies to mitigate false positives. Furthermore, most studies on honeyfiles concentrate on a single device, providing limited insights into attackers' characteristics, such as their interests. Due to the limitations of traditional honeyfiles, we have developed an enhanced version that considers all interconnected devices in the network, with a focus on improving the accuracy of insider attacker detection. Our research positions honeyfiles as a valuable tool for defensive deception. Unlike other honey assets like honeypots, honeyfiles are lightweight and require minimal maintenance. Deploying numerous honeyfiles can aid in identifying insider threats, but an excessive amount may negatively impact the defender and disrupt legitimate operations. Moreover, a high rate of false alarms can complicate the defender's assessment of system security. Therefore, it is crucial to assist defenders in determining the optimal number of honeyfiles to use.

#### Adaptive Honeyfile System Based on Bayesian Game

Research on honeyfiles has focused mainly on how the quantity of honeyfiles affects the efficacy of deception, but only with a static number of honeyfiles on a single device, which does not provide enough data for the defender to adjust its defense strategies. To address this gap, we present a novel honeyfile system, Mee, which employs centralized control and decentralized deployment to adjust the number of honeyfiles in the connected device based on the security assessment across the network. Unlike some existing deception research that only considers the interaction between attackers and defenders, we also discuss legitimate users in the scenario. Therefore, the defender needs to detect insider threats to reduce the system's risk and minimize the impact on legitimate users. We use the Bayesian game to model the interaction between insider attackers, legitimate users, and the defender. Our simulation results demonstrate that Mee can enhance accuracy when detecting insider attackers and reduce the impact on legitimate users.

#### Deep Reinforcement Learning Based Honeyfile System

Th previous project focuses on a relatively straightforward situation. Due to the restrictions of game theory, only two players are involved in the game at any given time, and many user and attacker behaviors are disregarded for simplicity. To make the model more realistic, we have employed deep reinforcement learning (DRL) for honeyfiles. The defender is treated as the agent that can detect misbehavior in the network through honeyfile alarms. The environment encompasses all connected devices, active users, and potential internal threats.

We present HoneyMee, a honeyfile system that considers all connected devices. It collects alarms when anyone interacts with honeyfiles and evaluates them, considering factors such as the typical behavior of legitimate users and the level of risk in the environment. HoneyMee is composed of HoneyMee Clients and a HoneyMee Controller. A HoneyMee Client serves as an endpoint application and keeps track of honeyfiles. It recognizes any potential file access on its local device, including any opening, editing, deleting, or transferring of a file. Whenever a honeyfile is accessed, the corresponding HoneyMee Client sends an alert to the HoneyMee Controller and awaits further instructions. The HoneyMee Controller is the defender of the network. It receives warnings from HoneyMee Clients and examines them to update its beliefs about the security level of each device. Upon receiving an alert, the controller determines whether the described access is by a legitimate user or an (insider) attacker based on its belief about the alarm history and the system security status.

We apply DRL to help the defender (the HoneyMee Controller) identify which honeyfile alarms indicate insider threats. The environment includes all connected devices, legitimate users, and insider attackers for the DRL scheme. The HoneyMee Controller's observations include the system security values and honeyfile alarms. This method goes beyond our previous approach to the problem, which was based on a Bayesian game and could tackle only one party (attacker or legitimate user) at a time. The DRL-based approach can handle a more realistic enterprise network and user models and supports multiple counterparties.

### 1.7.2 Research Approaches of Honeypots and GanFlow

Malicious reconnaissance (e.g., passive monitoring) is a critical component of an attacker's strategy to acquire information about a network and pinpoint valuable targets for infiltration. However, it is difficult to protect the system from such risks using traditional cybersecurity approaches, such as firewalls and intrusion detection systems, since they mainly concentrate on active threats and may not be as successful against passive monitoring techniques. Passive monitoring involves listening in on data transmissions without directly engaging with the system, making it more difficult to detect and prevent.

Defensive deception is a proactive cybersecurity strategy to protect valuable assets and data and to counter passive monitoring. It does this by introducing uncertainty and false information on what an attacker can observe. Deception technology can create false or misleading information that can confuse and deceive passive monitors, making it harder for attackers to obtain useful information. Furthermore, honeypots are purposely vulnerable systems or networks that are intended to draw in attackers. By deploying honeypots within a network, defenders can direct passive monitoring efforts towards fake assets, keeping the real ones safe. We present two approaches to help protect against passive monitoring attacks:

### Honeypot-Based Cyberdeception Against Reconnaissance via Hypergame Theory

Honeypots are a popular tool for cyberdeception, designed to confuse attackers and consume their resources and efforts. There are two varieties of honeypots, low-interaction and high-interaction, each with its own advantages and drawbacks. This research suggests a hybrid honeypot system that merges the two levels of honeypot complexity. High-interaction honeypots are more effective in deceiving knowledgeable attackers than low-interaction honeypots, which are simple to install and manage.

A two-player hypergame model is presented to determine how a defender should deploy low and high-interaction honeypots to protect the network from malicious reconnaissance activities. The tradeoff of each player is modelled and their optimal strategies are identified within a hypergame framework that takes into account the imperfect knowledge of each player about their opponent. Finally, numerical results are provided to demonstrate the effectiveness of the proposed honeypot system.

### GanFlow: Decoy Network Traffic Generation via Generative Adversarial Network

Generative Adversarial Networks (GANs) are generative modeling using deep learning methods, such as convolutional neural networks. The original GAN architecture includes *generator* and *discriminator*. Typically, a generator is trained to map from a latent space to data distribution, and the discriminator distinguishes candidates produced by the generator from the true data distribution. The objective of the trained generator is to increase the error rate of the discriminator. The generator network is employed to produce additional data in this latent variable space. As is typical for Generative Adversarial Networks, a discriminator network is trained to classify actual and generated packets from their latent representations.

We model the defender using the GAN generator, which attempts to learn the network flow patterns from actual servers and deceive the discriminator by generating realistic network packets. In addition, we use a discriminator to model the attacker's passive reconnaissance processes. We assume that an attacker can obtain the traffic from a compromised switch and identify the hosts' background information through the gathered data. However, the defender may use honeypot and honey traffic to confuse the attacker. We implement the GAN model, which includes a generator (i.e., the high-interaction honeypot) and a discriminator (i.e., the attack) within the CyberVAN testbed. The GAN generator learns network flow features from a server and produces honey traffic to confuse the attacker who performs passive monitoring via a compromised switch. Our simulation results show that the honey traffic can successfully disturb the attack's belief.

## 1.8 Organization

This dissertation is structured as follows:

- Chapter 2 provides an overview of existing defensive deception research based on game theory and machine learning. This survey categorizes and evaluates these studies based on their game theory or machine learning model, threat model, metrics, and simulation methods.
- Chapter 3 introduces two novel honeyfile systems to help defendants identify the optimal amount of honeyfiles to deploy and differentiate between insider attackers and legitimate users who have activated a honeyfile alarm. The integration of game theory and deep reinforcement learning techniques enables the defender to take advantage of environmental signals and make informed decisions for optimal outcomes.
- Chapter 4 introduces a two-player hypergame model which is presented to illustrate how a defender can deploy low- and high-interaction honeypots to protect a network from malicious reconnaissance activities. This model proposes a hybrid honeypot system.
- Chapter 5 discusses GanFlow, a network flow-based deception technique. This technique employs a Generative Adversarial Network (GAN) to learn from the patterns of real servers or hosts and guide honeypots to imitate these patterns to generate GanFlow, thus confusing attackers who are passively monitoring.
- Chapter 6 presents the conclusion of our research and answers the research questions posed in Section 1.6 of Chapter 1.

CHAPTER

# 2 A SURVEY OF ADAPTIVE DEFENSIVE DECEPTION

Research into defensive deception is becoming more and more popular, yet there has not been a thorough examination of its core elements, the principles that govern it, and the tradeoffs it presents in different scenarios. This survey concentrates on defensive deception research that is based on game theory and machine learning, as these are two of the most commonly used artificial intelligence techniques in defensive deception. This chapter provides an overview of the findings, lessons, and restrictions from prior studies. It outlines some research paths to address the major gaps in current defensive deception research.

In addition, we compare the existing deception techniques by considering different network environments, threat models, and experimental testbeds. We discuss the advantages and limitations observed from the deception techniques surveyed in this work. Based on these insights and limitations learned, we suggested promising future directions for defensive deception research.

**This chapter is joint work with:** Ahmed H. Anwar, Zelin Wan, Jin-Hee Cho, Charles Kamhoua, and Munindar P. Singh [Zhu et al., 2021b].

## 2.1 Motivation and Contribution

### 2.1.1 Motivation

Conventional security mechanisms, such as access control and intrusion detection, help deal with outside and inside threats but inadequately resist attackers who can subvert controls or launch new attacks. Deception is a distinct line of defense aiming to thwart potential attackers. The key idea of deception is to manipulate an attacker's beliefs to mislead their decision making, inducing them to act suboptimally.

Table 2.1: Comparison of our Survey with the Existing Surveys of Defensive Deception

| **Key Criteria** | **Our Survey (2020)** | **Lu et al. [2020] (2020)** | **Pawlick et al. [2019] (2019)** | **Han et al. [2018] (2018)** | **Rowe and Rrushi [2016] (2016)** | **Almeshekah and Spafford [2014] (2014)** |
|---|---|---|---|---|---|---|
| Provides concepts | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Provides key taxonomies | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Includes ML approaches | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ |
| Includes game-theoretic approaches | ✔ | ▲ | ✔ | ▲ | ▲ | ✘ |
| Describes attack types considered | ✔ | ✘ | ▲ | ▲ | ✔ | ▲ |
| Describes metrics | ✔ | ✘ | ▲ | ▲ | ▲ | ▲ |
| Describes evaluation testbeds | ✔ | ✘ | ✘ | ▲ | ▲ | ▲ |
| Is not limited to specific application domains | ✔ | ▲ | ✘ | ✔ | ✔ | ▲ |
| Discusses pros and cons of relevant techniques | ✔ | ▲ | ▲ | ✔ | ✔ | ▲ |
| Discusses insights, lessons, and limitations | ✔ | ▲ | ▲ | ✔ | ✔ | ▲ |
| Discusses future research directions | ✔ | ✔ | ✔ | ✔ | ▲ | ✘ |

✔: Fully addressed; ▲: Partially addressed; ✘: Not addressed at all.

Since the benefits of leveraging the core ideas of defensive deception (DD) have been realized in the cybersecurity research community, a good amount of research on DD has been explored.

Two main promising directions to develop DD techniques are observed in the literature. First, strategies of an attacker and defender have been commonly modeled based on game-theory where the defender takes DD strategies with the aim of creating confusion for attackers or misleading them so they would choose suboptimal or poor strategies. Second, DD techniques based on machine learning (ML) have been proposed to create realistic fake information or decoy objects that mimic real objects to mislead or lure attackers.

The synergistic merit of combining GT and ML has been recognized in the cybersecurity literature [Kamhoua et al., 2021], such as using game-theoretic defenses against adversarial machine learning attacks [Dasgupta and Collins, 2019; Zhou et al., 2019] or generative adversarial models for creating deceptive objects [Kamhoua et al., 2021]. However, little work has explored the synergies between GT and ML to formulate various cybersecurity problems. In particular, since players' effective learning of their opponents' behavior is critical to the accuracy of their beliefs of the opponents' types or next moves, using ML for the players to form their beliefs can contribute to generating optimal plays in a certain environment. In addition, when developing DD techniques, ML-based approaches can provide better prediction of attackers or high similarity in creating deceptive objects based on a large volume of data available. However, they may not provide effective strategic solutions under uncertainty which has been well explored in game-theoretic approaches. Therefore, this survey was motivated to facilitate future research taking hybrid DD approaches leveraging both GT and ML.

### 2.1.2 Comparison with Existing Surveys

We now distinguish the key contributions of our research from existing survey papers on DD techniques. Several studies have conducted surveys of DD techniques [Almeshekah and Spafford, 2014; Han et al., 2018; Lu et al., 2020; Pawlick et al., 2019; Rowe and Rrushi, 2016].

Almeshekah and Spafford [2014] described how defensive deception is considered in cybersecurity defense. To be specific, they discussed the following three phrases in considering a defensive deception technique: planning, implementing and integrating, and monitoring and evaluating. In particular, Almeshekah and Spafford [2014] discussed the models of planning deceptions in terms of affecting an attacker's perception which can be misled for a defender to achieve a system's security goals. However, their work is limited in its contribution to modeling and integrating DD to a limited set of attackers. In addition, this work did not consider the variety of network environments that should be considered in implementing DD techniques.

Rowe and Rrushi [2016] classified DD techniques in terms of impersonation, delays, fakes, camouflage, false excuses, and social engineering. They not only introduced the background on deception technologies but also explored the calculation of detectability and effectiveness of DD. However, their survey of game-theoretic DD is limited and lacks discussion of the state-of-the-art techniques.

Han et al. [2018] surveyed DD techniques based on four criteria, including the goal, unit, layer, and deployment of deception. They surveyed theoretical models used for DD techniques as well as the generation, placement, deployment, and monitoring of deception elements. Han et al. discussed the tradeoffs between various deceptive techniques in relation to whether they are deployed at the network, system, application, or data layers. Their discussion of game-theoretic deception, however, is not comprehensive.

Pawlick et al. [2019] conducted an extensive survey on DD taxonomies and game-theoretic DD techniques that have been used for cybersecurity and privacy. The authors discussed the six main types of deception categories: perturbation, moving target defense, obfuscation, mixing, honey-X, and attacker engagement. Their paper surveyed 24 papers published over 2008–2018, and defined related taxonomies to develop their own classification of game-theoretic DD techniques. This work is interesting in treating moving target defense and obfuscation as subcategories under DD. This survey discussed the common game-theoretic approaches used for developing DD techniques, such as Stackelberg, Nash, and signaling game theories. However, the survey and analysis of the existing game-theoretic defensive techniques conducted in this paper are limited to game-theoretical analysis without considering realistic network environments where ML-based DD techniques or a combination of these two (i.e., game theory and ML) may provide more useful insights and promising research directions.

Recently, Lu et al. [2020] conducted a brief survey based on the processes of DD consisting of three phases: planning, implementation and deployment, and monitoring and evaluation of deception. The authors discussed deception techniques based on information dissimulation to hide real information and information simulation to focus on attackers. This work briefly discussed game-theoretic DD and mainly focused on discussing the challenges and limitations of the current research. However, only a small fraction of the literature was included. In addition, this paper did not discuss ML-based DD approaches.

Some survey papers mainly focused on DD techniques for a particular type of attack or particular deception techniques. Carroll and Grosu [2011] investigated the effects of deception on the game-theoretic interactions between an attacker and defender of a computer network. They examined signaling games and related Nash equilibrium. However, this investigation only focused on honeypots technology while game-theoretic analysis of the deception is limited in examining the interactions between an attacker and a defender in the signaling games. Virvilis et al. [2014] surveyed partial DD techniques that can be used to mitigate Advanced Persistent Threats (APTs).

In Table 2.1, we summarized the key contributions of our survey paper, compared to those of the five existing survey papers [Almeshekah and Spafford, 2014; Han et al., 2018; Lu et al., 2020; Pawlick et al., 2019; Rowe and Rrushi, 2016] based on several key criteria.

### 2.1.3 Key Contributions

In this chapter, we made the following **key contributions**:

1. We provided a novel classification scheme that characterizes a DD technique in its conceptual deception categories, presence of an object (i.e., physical or virtual), expected effects after applying deception, the ultimate goal (i.e., for asset protection or attack detection), and activeness (i.e., active or passive, or both). This provides an in-depth understanding of each deception technique and insights into how it can support a system's security goal.
2. We discussed key design principles of DD techniques in terms of what-attacker-to-deceive, when-to-deceive, and how-to-deceive. In addition, based on the key properties of DD techniques, we identified the key benefits and caveats when developing DD techniques leveraging game theory (GT) and ML.
3. We discussed GT and ML-based DD techniques based on their types along with pros and cons. In addition, using the classification scheme in Section 2.2, we discussed both GT and ML algorithms.
4. We surveyed attacks handled by the existing game-theoretic and ML-based DD techniques. Accordingly, we discussed what attacks are more or less frequently considered in the literature by the DD techniques.
5. We surveyed how DD techniques are mainly considered to deal with the challenges of different network environments as application domains and discussed the pros and cons of the deployed game-theoretic or ML-based DD techniques.
6. We examined what types of metrics and experiment testbeds are more or less frequently used in game-theoretic or ML-based DD techniques to prove their effectiveness and efficiency.
7. We extensively discussed lessons and insights learned and limitations observed from the DD techniques surveyed in this work. Based on these insights and limitations learned, we suggested promising future directions for DD research.

Note that the scope of this chapter is mainly focused on surveying GT or ML-based DD techniques and discussing insights, limitations, or lessons learned from this extensive survey. Hence, some DD techniques that do not use either GT or ML are excluded in this survey.

### 2.1.4 Research Questions

We address the following **research questions** in this chapter.

**RQ Characteristics:** What key characteristics of DD distinguish it from other defensive techniques?

**RQ Metrics:** What metrics are used to measure the effectiveness and efficiency of the existing game-theoretic or ML-based DD techniques?

**RQ Principles:** What key design principles help maximize the effectiveness and efficiency of DD techniques?

**RQ GT:** What are the key design features when a DD technique is devised using GT?

**RQ ML:** What are the key design features when a DD technique is developed using ML?

**RQ Applications:** How should different DD techniques be applied in different application domains?

We answered these questions in Section 2.9.1.

### Structure of This Chapter

The rest of this chapter is structured as follows. Section 2.2 provides the concept of deception and taxonomies related to defensive deception (DD). Section 2.3 discusses the key principles of designing a DD technique. In addition, this section clarifies the key distinctive characteristics of the DD techniques, compared to other defense techniques that achieve the same defense goal. Section 2.4 explains the key components of using game-theoretic DD and surveys the existing game-theoretic DD techniques along with discussions of their pros and cons. Section 2.5 discusses the key components of leveraging ML techniques to develop DD techniques. In addition, this section extensively surveys the existing ML-based DD techniques and addresses their pros and cons. Section 2.6 describes attack types countered by the existing game-theoretic and ML-based DD techniques. Section 2.8 presents metrics to measure the effectiveness and efficiency of the existing DD techniques using game theory (GT) and ML. In addition, this section surveys evaluation testbeds used for validating those existing DD techniques surveyed in this work. Section 2.7 discusses how GT or ML-based DD techniques have been developed for different application domains, such as enterprise networks, cyberphysical systems (CPS), cloud web-based networks, Internet of Things (IoT), software-defined networks (SDNs), and wireless networks. Section 2.9 summarizes the insights and lessons learned by answering the key research questions raised in Section 2.1.4. In addition, this section discusses the limitations found from the DD techniques surveyed in this work and suggests promising future research directions.

## 2.2 Taxonomies of Defensive Deception

Deception has been heavily used by attackers who perform a variety of attacks at different levels of applications in both computer and social systems. In this section, we limit our discussions of deception in terms of a defender's perspective. This section mainly discusses formal models of deception, related common taxonomies used in developing defensive deception techniques, and their distinctive characteristics.

### 2.2.1 Formal Models of Cyberdeception

In this section, we discuss formal models of cyberdeception that have been discussed in the literature. Although there are numerous formal models of cyberdeception, we limit our discussion to formal modeling approaches based on logic, probabilistic logic, and hypergame theory.

### Logic-based Cyberdeception

Deception planning has been studied based on logical reasoning of the key components of deception and their states (i.e., status). Deception modeling logic [Jafarian and Niakanlahiji, 2020; Takabi and Jafarian, 2017] was proposed to provide a formal deception model, $\mathcal{DM}$, as:

$$\begin{aligned} \mathcal{DM} = \{&\text{facts, beliefs, actions,} \\ &\text{causation rules, attackers, goal, budget}\} \end{aligned} \tag{2.1}$$

Each component of the DM is:

- *Facts*: The facts indicate a deception problem that can be described by (1) *attributes* of a system (e.g., configuration parameters, such as an operation system of a host, a server's criticality, existing routes between hosts), (2) data types of each attribute (e.g., Boolean, ordered, integral), and (3) actual values assigned to the attributes.
- *Beliefs*: These indicate an attacker's beliefs on the set of attributes, which may not be aligned with the true states of the attributes.
- *Actions*: An action refers to a defender's action to manipulate the attacker's belief on an attribute. Attributes can be *actionable* or *derivative*. Actionable attributes can be directly manipulated to influence the attacker's belief by taking a deceptive action. Derivable attributes cannot be directly manipulated but can be indirectly derived from the attacker's beliefs over other attributes. When modeling a set of actions, each action should be plausible to the attacker so the deception action is promising enough to deceive the attacker. In addition, the action should consider its cost.
- *Causation rules*: The causation rules apply to each attribute to determine what (e.g., facts or actions) can cause the manipulation of an attacker's belief on the attribute.
- *Attackers*: A set of attacker types can be modeled based on the probability distribution over the set when the probability of an attacker's particular type is available.
- *Goal & Budget*: The concrete benefit of successful or failed deception should be estimated in terms of defense effectiveness, defense cost (including both monetary budget), expected effect (i.e., a required level of achieved system performance/security) or other associated impact (e.g., interoperability with other defense or service mechanisms).

Rowe [2007] proposed a deception planning method to systematically provide logically consistent deceptions when users are detected as suspicious or malicious by an intrusion detection system. The attack modeling technique introduced by Rowe [2007] takes into account authentic attackers defined by their targets (such as files, a network, statuses like log-in or administrator privileges), the conditions of the targets (like presence, permission, preparedness, or functionality), and their attributes (like file size, network capacity, multiple locations, or password length). By using an inference rule based on the status, an attacker can derive the status of a targeted resource. To prevent the attackers from achieving their goal, deception tactics can be used to deny the service the attacker requested by generating error messages or respond with a credential request (e.g., password or other credential information). An error message can be generated in the form of *a file not transferable*, *a file not recognizable*, or *protection violation*.

### Probabilistic Cyberdeception

This type of formal models mainly studies how a developed DD technique can manipulate an attacker's belief and accordingly influence its action based on its belief in a probabilistic manner. Jajodia et al. [2017] proposed a probabilistic logic model for cyberdeception. To maximize the probability of a successful deception, the defender decides to send true or fake responses to the attacker scanning probes. The effectiveness of deception is measured in terms of the accuracy of the attacker's belief on the system attributes (i.e., how much information the attacker has obtained towards a target system accurately). Crouse et al. [2015] proposed probabilistic models of reconnaissance-based defense with the aim of providing in-depth understanding on the effect of the defense on system security. The authors quantified attack success under various network conditions in terms of a network size, a size of deployment, and several vulnerable system components. The proposed deception models provide performance analysis of probabilistic triggering of network address shuffling and deployment of honeypots when these defense mechanisms are considered separately or in concert. In prior work [Cho et al., 2019], we used Stochastic Petri Nets to model an attack-defense hypergame with the aim of examining how an attacker's beliefs manipulated by a DD can affect its decision making, resulting in attack failure. As Crouse et al. [2015]; Jajodia et al. [2017] and Cho et al. [2019] observe, probabilistic models of deception games can effectively capture dynamic interactions between an attacker and a defender under uncertain environments.

### Cyberdeception Hypergames

Much DD research uses game theory to formulate an attack-defense game. In particular, to effectively deal with uncertainty in real scenarios, a sequential attack-defense game where the defender uses DD has been formulated based on hypergame theory. Ferguson-Walter et al. [2019b] formulated a cyberdeception game as a sequential game as $G = (\mathcal{P}, \mathcal{M}, \Theta, u, \mathcal{T})$, where $\mathcal{P}$ refers to a set of players, $\mathcal{M}$ refers to a set of actions, such as $\mathcal{M} = \{\mathcal{M}_i\}$ for player $i$, $\Theta$ is a set of strategies, $\Theta = \{\Theta_i\}$ for player $i$, $u$ is a utility function, $u = \{u_i\}$ for player $i$, and $\mathcal{T}$ is a sequence of moves when players take turns in sequence. By using the formulation of a regular, sequential game, a cyberdeception can be formulated as $(G, G^A, G^D)$ where $G^A$ and $G^D$ are derived games. In $G$, a move history of an attacker, $\mathcal{M}^A = \{m^A\} = \{m^A_1, m^A_2, \ldots, m^A_r\}$, and a defender, $\mathcal{M}^D = \{m^D\} = \{m^D_1, m^D_2, \ldots, m^D_s\}$, respectively, refers to the sequence of moves in the game where $r+s \leq \mathcal{T}$. For simplicity, the move histories of the attacker and defender can be denoted by $m = \{m^A, m^D\}$. Given that $\mathcal{M}^{X|Y} = m^{X|Y}$ refers to $Y$'s perception of the set of $X$'s actions (i.e., it is not necessarily true for $\mathcal{M}^{X|Y} = \mathcal{M}^X$), the attacker (A)'s beliefs about the move sequence, $m$, can be represented by $m^{*|A} = \{m^{A|A}, m^{D|A}\}$. Similar to the notation of the moves perceived by A, we can also obtain $u^{*|A} = (u^A|A, u^D|A)$, which is A's perception of $u = \{u^A, u^D\}$ and $\Theta^{*|A} = \{\Theta^{A|A}, \Theta^{D|A}\}$, which is A's perception of sets of strategies for $\Theta^A$ and $\Theta^D$. Hence, the derived game can be denoted by $G^A = \{\mathcal{P}, \mathcal{M}^{*|A}, u^{*|A}, \mathcal{T}\}$, which is A's perception towards the game $G$, and $G^D = \{\mathcal{P}, \mathcal{M}^{*|D}, u^{*|D}, \mathcal{T}\}$, which is D's perception towards the game $G$. If an attack-defense game does not use the concept of hypergame, it simply relies on the game formulation of $G$.

## 2.2.2 Concepts of Defensive Deception

The conventional concept of military deception refers to actions taken to intentionally mislead an enemy about one's strengths and weaknesses, intents, and tactics [Sharp, 2006]. The defender employs deception to manipulate the enemy's actions to advance one's mission [Sharp, 2006]. Defensive deception (DD) applies to a wide spectrum of interactions between an attacker and a defender under conflict situations [Sharp, 2006]. The concept of DD initiated in military environments has been introduced to cyber domains with the name of cyberdeception. Almeshekah and Spafford [2016] refined the concept of cyberdeception in [Yuill, 2006] as "planned actions taken to mislead and/or confuse attackers and to thereby cause them to take (or not take) specific actions that aid computer-security defenses." Although the concept of deception is highly multidisciplinary [Guo et al., 2020a], the common idea is to mislead an entity so it forms a false belief and to control its behavior based on that false belief [Rowe and Rrushi, 2016]. Therefore, when deception is successfully executed, a deceivee responds suboptimally, which provides benefits for a deceiver and results in achieving the deceiver's goal. Rowe and Rrushi [2016] emphasized manipulation as the core concept of deception where deception encourages a deceivee to do something a deceiver wants or discourages the deceivee from doing something the deceiver does not want.

As the discussion of deception in this work is limited to DD, we consider a defender's deception against an attacker to achieve the defender's goal.

Table 2.2: Taxonomies and Classification of Defensive Deception Techniques

| **Deception Tactic** | **Presence of a Real Object** | **Expected Effect** | **Ultimate Goal** | **Activeness** |
|---|---|---|---|---|
| Masking | True | Blending, Hiding, Misleading | Asset Protection | Passive |
| Repackaging | True | Blending, Hiding, Misleading | Asset Protection | Passive |
| Dazzling | True | Hiding, Confusing, Misleading | Asset Protection | Passive |
| Mimicking | False | Luring, Misleading | Attack Detection | Passive |
| Inventing | False | Luring, Misleading | Attack Detection | Active |
| Decoying | True | Luring, Misleading | Attack Detection | Active |
| Bait | True | Luring, Misleading | Attack Detection | Active |
| Camouflaging | True | Blending, Hiding, Misleading | Asset Protection | Passive |
| Concealment | True | Hiding, Misleading | Attack Detection | Active |
| False Information | False | Luring, Confusing, Misleading | Asset Protection or Attack Detection | Active or Passive |
| Lies | False | Hiding, Misleading | Asset Protection | Passive |
| Display | True | Hiding, Misleading | Asset Protection | Passive |

### 2.2.3 Taxonomies of Defensive Deception

Deception can be applied with certain intent, either malicious or nonmalicious intent [Guo et al., 2020b]. In particular, deception with malicious intent has been used as an attacker's strategy to deceive a defense system. For example, spammers are paid from clicking deceptive advertisements [Nextgate, 2019], malicious users disseminate phishing links to obtain credentials from victim users [Vergelis et al., 2019], or social and political bots propagate false information to influence public opinions [Forelle et al., 2015].

As we consider deception in the context of DD, we limit our discussions to deception *with intent* where the intent is to defend a system against attackers. In this section, we discuss taxonomies in terms of the following aspects: (i) what conceptual techniques are used; (ii) whether a deception uses a true object or not (i.e., a true object exists but is hidden to deceive or a fake object is created to deceive); (iii) what effects are expected when successfully deceiving an opponent; (iv) what is an ultimate goal of the deception, such as detecting an attacker or protecting a system; and (v) whether a deception is active (mainly for attack detection) or passive (mainly for attack protection) to deceive an opponent.

#### Conceptual Deception Tactics

In [Almeshekah and Spafford, 2016; Bell and Whaley, 1991; Bennett and Waltz, 2007; Dunnigan and Nofi, 2001], we found the following conceptual deception techniques:

- *Masking* [Almeshekah and Spafford, 2016; Bell and Whaley, 1991]: This refers to hiding certain information or data in the background.

- *Repackaging* [Almeshekah and Spafford, 2016; Bell and Whaley, 1991]: This means to hide a real object as something else to hide the real object. A defender can make a vulnerability look like something else. For example, a defender can create honey files [Yuill et al., 2004], such as creating trap files that look like regular files.

- *Dazzling* [Almeshekah and Spafford, 2016; Bell and Whaley, 1991]: This is a different way to hide something that is hard to blend in with the background or to be repackaged as something else. A real object can be hidden by overshadowing it, aiming to be undetected as the real, true object. For example, an intruder can be confused by receiving many error messages.

- *Mimicking* [Almeshekah and Spafford, 2016; Bell and Whaley, 1991]: This refers to imitating aspects of a real object. This is often used to look honeypots as real nodes or interfaces to attackers.

- *Inventing* [Almeshekah and Spafford, 2016; Bell and Whaley, 1991]: A defender can create a new fake object to lure attackers. For instance, a software can be presented in a honeypot for attackers to download it, which enables collecting the attackers' personal data.

- *Decoying* [Almeshekah and Spafford, 2016; Bell and Whaley, 1991]: A deceiver attracts an attacker's attention away from critical assets that should be protected in a system. For example, the deceiver can provide publicly available false information about the system configurations to hinder reconnaissance attacks.

- *Bait*: Deception can use truth [Bennett and Waltz, 2007] to earn trust from a deceiver. That is, a defender can present attackers with correct but low-grade information to lure them. The truth can be the information that an attacker already has, or the deceiver needs to sacrifice some correct sensitive data to effectively lure attackers.

The military literature provides the following taxonomy for cyberdeception [Dunnigan and Nofi, 2001].

- *Concealment* [Dunnigan and Nofi, 2001]: This deception is similar to hiding in the classical taxonomy. For the defensive purpose, it helps for honeypots to conceal their user-monitoring software so they look more like normal machines.
- *Camouflage* [Dunnigan and Nofi, 2001]: This deceives an attacker by blending a real object into a background or environment. For example, valuable files and programs can be assigned misleading names to make it difficult for an attacker to find them.
- *False information* [Dunnigan and Nofi, 2001]: Fake information (or disinformation) can be planted to mislead an attacker in cyberspace. However, most false information about cybersystems is easy to be checked by testing or trying it out. Therefore, this kind of deception cannot fool attackers for a long time.
- *Lies* [Dunnigan and Nofi, 2001]: This provides deceptive information, which is false. However, lies are distinguished from disinformation because they are often provided as responses to questions or requests. In this sense, even if this deception uses false information, it implies a passive action because lies may not be used unless it is requested. Lies could be more effective than explicitly denying access as they encourage an attacker to continue wasting time trying to access the resource in different ways and at different times.
- *Displays* [Dunnigan and Nofi, 2001]: This deception is similar to *inventing* as it also aims at misleading objects or information. However, unlike the inventing, which creates a new, fake object, displays can be applied to a real object (i.e., an object that is present in the system) by displaying it differently.

### Presence of Actual Objects or Information

Deception can be applied when an actual, true object exists or no actual object exists. To be more specific, a defender can use true objects or information but may want to hide it by lying or providing false information towards it. For example, even if a system uses Windows as an operating system but may lie it uses Unix. But the system is using a certain OS in deed. On the other hand, a fake object, which has not existed anywhere, can be created for the purpose of creating uncertainty or confusion, which can lead an opponent to reach a suboptimal or poor decision, such as honey tokens or honey files.

### Expected Effects (or Intents) of Defensive Deception

Via defensive deception (DD), one or more effects are expected as the process of achieving the goal of deception, making an attacker choose a suboptimal choice in its strategies. We categorize these in the following five aspects:

- *Hiding*: This effect is addressed when deception is used to change the display of a real object into something else. Some obfuscation defense techniques also use hiding data to slow down the escalation of attackers or increase attack complexity [Almeshekah and Spafford, 2014; Rowe and Rrushi, 2016].
- *Luring* [Almeshekah and Spafford, 2016; Bell and Whaley, 1991]: This effect is shown when a fake object is demonstrated to an attacker where the object looks like a real object an attacker is interested in. The typical examples are honeypots to lure attackers or honey files to send false information to the attackers.
- *Misleading* [Daniel and Herbig, 1982]: Most deception techniques have this effect as the last step to achieve the ultimate goal of deception, misleading an attacker to choose a suboptimal action.
- *Blending*: This effect is made when a real object can be well blended into background or environments. Its effectiveness relies on an environment and in altering the appearance of things to hide. For instance, give password files a nondescriptive name to elude automated searches run by hackers looking for files named 'pass' [Yuill et al., 2006]. Additionally, blending hidden objects with the environment can be achieved through altering the environment instead. For example, a defender can create noise in the environment through adding bogus files to make it harder to find critical ones.
- *Confusing* [Almeshekah and Spafford, 2014]: This effect comes from perceived uncertainty caused by a lack of evidence, conflicting evidence, or failing in discerning observations (e.g., cannot pinpoint whether a car on the street is red or blue) [Almeshekah and Spafford, 2014; Jøsang et al., 2018]. A defender can confuse an attacker by presenting both truths and deceits [Almeshekah and Spafford, 2014].

Some deception techniques use more than one technique, such as bait-based deception including lies to increase attack cost or complexity [Keromytis and Stolfo, 2014]. Bowen et al. [2009] designed a trap-based defense technique to increase the detection of an insider attack using several deception techniques like camouflage, false information, and creating fake objects. First, the deception system embeds a watermark in the binary format of the document file to detect when the decoy is loaded. Moreover, the content of each decoy document includes several types of "bait" information, such as online banking logins, login accounts for online servers, and web-based email accounts. A "beacon" is embedded in the decoy document that signals a remote web site upon opening of the document.

### Ultimate Goals of Defensive Deception in Cyberspace

The ultimate goal of a defender using deception techniques is either protecting assets or detecting attackers or both. Although some deception techniques allow the defender to achieve both asset protection and attack detection, deception by hiding is more likely to protect system assets while deception by using false objects or information is to catch attackers.

### Activeness of Defensive Deception

Caddell [2004] categorized deception as *passive* vs. *active*. *Passive deception* uses hiding valuable assets or their capabilities and information from attackers. On the other hand, *active deception* uses fake, false information aiming for the attacker to form false beliefs, leading to making suboptimal decisions. Even if false information is used, the defense goal can be either protecting assets or detecting attackers. Hence, the distinction between active and passive deception may not always be clear.

In Table 2.2, we summarized how a particular deception technique can be understood based on the four key aspects: the presence of actual objects or information used in deception, expected effects (or intents), the ultimate goal(s), and the activeness (or passiveness) of the used deception technique.

## 2.3 Design Principles and Features of Defensive Deception

In this section, we discuss the four key design principles of DD techniques. In addition, we address the unique properties of DD and their key merits and caveats when using DD techniques. Further, we discuss how the DD techniques differ from other similar defense techniques, such as moving target defense or obfuscation techniques.

### 2.3.1 Design Principles of Defensive Deception

In this section, we discuss the design principles of DD in terms of the four aspects: *what-attacker-to-deceive* (i.e., what type of an attacker to deceive), *when-to-deceive* (i.e., when deception can be used), *how-to-deceive* (i.e., what particular deception technique can be used to deceive an attacker). We discuss each principle as follows:

#### What-Attacker-to-Deceive

This design principle asks to determine what type of an attacker a defender wants to deceive. For example, if the defender targets to deceive attackers performing reconnaissance attacks as outside attackers, it may aim to deceive them by providing false information about a system configuration (e.g., saying it uses Windows operating system (OS) even if it actually uses Unix). In addition, if a valuable system asset should be protected from attackers aiming to exfiltrate confidential information to an outside network, the defender may deploy a honeypot that mimics a real node with highly confidential information (e.g., a database). Hence, determining *what-attacker-to-deceive* is to decide what attackers to target by the defender. Since developing a DD technique incurs a cost, there should be in-depth analysis and investigation into whether a specific attacker should be prevented or detected by a DD technique in terms of cost, effectiveness, and efficiency of the technique's deployment.

#### When-to-Deceive

This design principle refers to determining when a deception technique should be used in terms of the attack stage of an attacker in the cyber kill chain (CKC) [Chen et al., 2014; Okhravi et al., 2013]. The six CKC stages include reconnaissance, delivery, exploitation, command and control, lateral movement, and data exfiltration [Okhravi et al., 2013]. Outside attackers mainly perform attacks in the stage of reconnaissance and delivery stages with the aim of penetrating a target system. On the other hand, inside attackers aim to perform attacks to exfiltrate confidential information to the outside, unauthorized parties. For example, when more malicious scans are identified in the reconnaissance stage, the defender can use fake patches with false vulnerability information. As a result, this can mislead the attacker to target honeypots. After the attacker successfully penetrates the system as an inside attacker, the defender may use honey files or tokens to deceive the attacker to exploit them. However, there are also legacy defense mechanisms, such as access control, intrusion detection, and prevention, or emerging technologies, such as moving target defense as the alternative mechanisms that deal with the same types of attackers. In such cases, utility analyses should be based on losses and gains in terms of the timing of using a DD technique.

#### How-to-Deceive

The design decision on how-to-deceive in employing DD is related to what type of DD technique to use. The conceptual deception techniques have been discussed in Section 2.2.3, such as masking, mimicking, decoying, false information, baits, and so forth. However, what specific technique to use for DD is more related to what technology to use to achieve deception. Some example DD technologies include honeypots, honey files, honey tokens, fake patches, fake network topologies, fake keys, or bait files [Han et al., 2018].

### 2.3.2 Distinctions between Defensive Deception and Other Similar Defense Techniques

Defensive deception (DD) is often mentioned as moving target defense or vice versa. In addition, obfuscation has been used to achieve the same aim as DD. In this section, we discuss how the roles and natures of these three techniques differ from each other, as well as how they overlap in their functionalities and aims.

#### Distinction with Moving Target Defense (MTD)

MTD is similar to DD in terms of its aim to increase confusion or uncertainty of attackers, leading to deterring the escalation of their attacks to the next level or failing their attacks. However, the key distinction is that MTD does not use any false information to actively mislead attackers while DD often involves using false objects or information for the attackers to form false beliefs and be misled to make suboptimal or poor attack decisions. MTD relates its key functionality mainly with how to change system configurations more effectively and efficiently while DD involves the exploitation of manipulating the attacker's perception. If DD is well deployed based on a solid formulation of manipulating the attacker's perception, it can be more cost-effective than deploying MTD. Ward et al. [2018] considered MTD as part of DD while Cho et al. [2020] treated DD as part of MTD. The distinction between these two is not clear because DD can be deployed using MTD techniques, such as randomness or dynamic changes of system configurations (e.g., dynamically assigning decoys in a network) while it may not use false information all the time (e.g., baits or omission such as no response).

#### Distinction with Obfuscation

Obfuscation techniques have been used by an attacker to obfuscate malware [You and Yim, 2010], metamorphic viruses [Borello and Mé, 2008], or malicious JavaScript code [Xu et al., 2012]. In terms of a defender's perspective, obfuscation techniques have been used to obfuscate private information [Ardagna et al., 2007], or Java bytecode for decompiling [Chan and Yang, 2004]. Although obfuscation techniques are often mentioned as part of DD techniques, they have been studied as a separate research area for decades. Chan and Yang [2004] conducted an extensive survey on obfuscation techniques

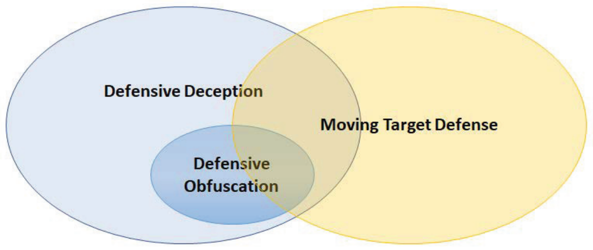


Figure 2.1: Relationships between defensive deception, moving target defense, and defensive obfuscation.

enhancing system security and discussed the key aims of obfuscation techniques as: (i) increasing the difficulty of reverse engineering of the program logic; (ii) mitigating the exploitability of vulnerabilities by attackers; (iii) preventing modifications by unauthorized parties; and (iv) hiding data or information. Although the concept of DD has gained popularity of late while obfuscation techniques have been applied for decades, it is obvious that the aim of defensive obfuscation is well aligned with that of DD. However, a subtle difference lies in that DD involves cognitive aspects of an attacker's perception while defensive obfuscation is directly involved with achieving security goals (e.g., confidentiality, data integrity, availability). Defensive obfuscation techniques are also used in combination with diversification [Chan and Yang, 2004] (e.g., diversifying systems, network configurations, or components), which is a well-known concept used by MTD techniques.

Based on our understanding on the functionalities and aims of DD, MTD, and defensive obfuscation, we would like to hold our view based on Fig. 2.1, showing the relationships among them. Our view is that a defensive obfuscation can belong to either MTD or DD. In addition, there are also an overlapping area of MTD and DD where some defensive techniques require using the features of both concepts.

## 2.4 Game-Theoretic Defensive Deception (GTDD)

In this section, we discuss the key component of modeling DD techniques using game theory. In addition, we discussed a wide range of DD techniques that have been considered based on various types of game theory with their pros and cons.

### 2.4.1 Key Components of Game-Theoretic Defensive Deception

Game theory has been used extensively to solve cybersecurity problems by modeling the following key elements: a defender and attacker as players, actions as attack and defense strategies, observability of a system and an opponent's strategies, or system dynamics. We discuss these key elements of cybersecurity games as follows.

#### Players

Most research taking game-theoretic DD approaches model a two-player game where the two players are an attacker and a defender using DD [Kiekintveld et al., 2015; Mao et al., 2019; Pawlick and Zhu, 2015]. However, even in a two-player game, some deception games modeled different types of players under each player. Pawlick and Zhu [2015] modeled a two-player signaling game where a defender as a sender can be either a normal system or a honeypot while an attacker as a receiver has one type. Depending on the type, the defender takes its strategy. Pawlick et al. [2015] introduced a three-player game between a cloud defender, an attacker and a device which is connected to the cloud where the defender can send false signals to deceive the attacker.

### (In) Complete Information

A complete information game assumes players have full knowledge of the game parameters, such as possible actions to be taken by other players, a reward function, and the current state of the game in case of dynamic or multistage games. A game with complete information is sometimes considered to be impractical. The assumption of the defender knowing the set of attack actions to be taken by its attacker is not realistic due to inherent uncertainty in a context. On the other hand, an incomplete information game represents a practical class of games where one or more players may or may not know some information about the other players [Kajii and Morris, 1997]. For instance, a player may not fully know other players' types, strategies, and payoff functions.

### (Im) Perfect Information

An imperfect information game represents a game in which a player may not know what exact actions have been played by other players in the game. This makes it computationally prohibitive to track the history of actions in case of a multistage game. However, all players may know their opponents' types, strategies, and payoff functions, forming a complete, imperfect information game. This imperfect information game is often assumed in cybersecurity games between an attacker and a defender [Başar and Olsder, 1999; Phlips, 1988].

### Partial Observability

Players acting upon dynamic systems are modeled as dynamic and stochastic games. In this case, the system/environment state changes over time and depends on the actions taken by all the players. However, in some scenarios, one or more players will not fully observe the state. This results in a partially observable game. If the system environment is affected by an exogenous factor, the state transition is stochastic, resulting in a partially observable stochastic game (POSG) [Hansen et al., 2004]. In the special case of POSG with only a single player, the game turns to a partially observable Markov dynamic process [Cassandra, 1998]. POSG model deals with the most general form of games that capture different game settings, such as incomplete information as well as imperfect information (i.e., imperfect monitoring). Therefore, solving such a game efficiently is still an open research question.

### Bounded Rationality

Both the rational and bounded rational players are commonly applied in a game between an attacker and a defender. A rational player can always choose an optimal strategy to maximize its expected utility. However, a bounded rational player has only limited resources and cannot afford an unlimited search to find an optimal action [Tadelis, 2013].

**Pure or Mixed Strategies**

Two main strategies in game theory are pure strategy or mixed strategy [Tadelis, 2013]. A pure strategy means a player's strategy is determined with the probability 0 or 1. In contrast, a mixed strategy is a probability distribution of several pure strategies. For example, Clark et al. [2012] used pure strategies to perform deceptive routing paths to defend against jamming attacks in a two-stage game using Stackelberg game theory. Zhu et al. [2012] also used deceptive routing paths against jamming attacks but identified an optimal routing considering resource allocation based on mixed strategies.

**Uncertain Future Reward in Utility**

Due to the inherent uncertainty caused by multiple variables that may control the future states, a player's utility cannot be guaranteed [Tadelis, 2013]. In game-theoretic DD research, uncertainty is mainly caused by nonstationary environmental conditions and nondeterministic movements (or actions) by opponent players. In a signaling game, uncertainty is caused by both the movement of an opponent player and its type [Mohammadi et al., 2016; Pawlick and Zhu, 2015] where a player can take an action after it knows its opponent's type.

**Utilities**

For a player to take an optimal action, it is critical to know both its own payoff (or utility or reward) function and its opponent's payoff function. In a static, one-time game, a reward per strategy is based on action profiles (i.e., a set of actions each player can take) of all players [Tadelis, 2013]. However, in a sequential game (i.e., a repeated game), a reward can be estimated based on the history of action profiles from the beginning to the end of the game. Hence, in such games, a player aims to maximize the expected average or accumulated reward over the period of the sequential game [Başar and Olsder, 1999].

**Full Game or Subgame**

A full game represents a game where each player considers a set of all possible actions to select one action to maximize its utility. A subgame indicates a subset of the full game where a player considers a subset of all possible actions when selecting one action to maximize its utility. Particularly, in sequential games, a subgame includes a single node and all its successors [Tadelis, 2013]. It is similar to a subtree in a tree-structure sequential game, which refers to an extensive form game. House and Cybenko [2010] used the hypergame theory to model the interactions between an attacker and a defender. Hypergame uses a subgame to model each player's different view on the game and corresponding strategies under the subgame. In addition, Zhang et al. [2019] and Xu and Zhuang [2016] used *Subgame Perfect Nash Equilibrium* (SPNE) [Mas-Colell et al., 1995] (i.e., a Nash equilibrium in every subgame) to deal with the resource allocation for both the defender and attacker in non-hypergames.

**Static Game vs. Dynamic Game**

In game theory, a static game is considered as one-time interaction game where players move simultaneously. Hence, a static game is a game of imperfect information. On the other hand, a dynamic game is considered as a sequential game assuming that players interact through multiple rounds of interactions as repeated games. Although most dynamic games are considered sequential games with perfect information, if a game considers partial (or imperfect) observability, the dynamic game can be sequential across rounds of games but within a subgame, it can be a game of imperfect information (i.e., players know other players' moves in previous rounds but do not know their moves in a current round).

Game theory has provided a powerful mathematical tool for solving strategic decision making in various dynamic settings. Since the powerful capability of game theory has been clearly proven based on the achievement of several game theorists as Nobel Prize Laureates [Tadelis, 2013], there is no doubt that game theory has substantially inspired methods for solving complex, dynamic decision-making problems to produce mathematically validated solutions. Although game theory has been substantially explored to provide solutions with solid theoretical proofs, one of its common limitations has been discussed in terms of too strong assumptions to derive Nash Equilibria, such as common knowledge about probability distributions of Nature's moves or players' correct beliefs towards opponents' moves. As game theory has a long history since 1700s [Bellhouse, 2007], its long evolution has made significant advancement in considering increasingly realistic aspects of real-world problems. The exemplary efforts in addressing realistic scenarios include the consideration of bounded rationality, decision making under uncertainty using the concept of mixed strategies, games of imperfect or incomplete information, hypergame with different subjective views of players, games with partially observable Markov Decision Process (POMDP), subjective beliefs, subjective rationality, and so forth [Tadelis, 2013]. We believe these efforts have obviously opened a door for game theory to be highly applicable in real systems, including systems concerning cybersecurity. In the context of developing DD techniques, game theory can contribute to manipulating an attacker's beliefs and accordingly leading the attacker to miscalculate utilities under uncertainty with the aim of failing the attacker. In addition, how frequently a certain defense should be executed or what defense strategy to choose can be also determined based on game-theoretic decision methods.

### 2.4.2 Common Games Formulated for Defensive Deception

We now discuss popular game theories used to formulate various DD techniques. We will discuss the following games:

- *Bayesian games*: A Bayesian game is a game of incomplete information which means part of or all players do not know their opponents' types, actions, or payoffs where each player knows their own type, a set of actions, and corresponding payoffs. Hence, each player has a subjective prior probability distribution of their opponent's type [Harsanyi, 1967].

- *Stackelberg games*: Stackelberg game refers to a sequential two-player game where the first player is a leader and the second player is a follower. In a stage game, the leader moves first, then the followers observe the action of leader and select their strategy based on the observation [Von Stackelberg, 2010]. Hence, the first player takes the first-mover-advantage which can lead the second player's move to their intended direction [Tadelis, 2013].

- *Signaling games*: This is a sequential Bayesian game where one player sends a signal (i.e., information) to another player. When the player sends false information, it can be most costly [Tadelis, 2013].

- *Stochastic games*: This is a multistage game in which the game evolves from one stage (state) to another following a stochastic distribution. The transition probability matrix depends on a set of action profiles of both players and may also depends on random exogenous factors of the game environment. The game dynamics determine the payoffs of both players [Shapley, 1953].

- *Games with Partially Observable Markov Decision Process (POMDP)*: The POMDP considers to model an agent's decision process where the agent selects actions to maximize a reward of the system. However, the agent cannot directly perceive the system state, but it can obtain observations depending on the state. The agent can maintain a belief of a system state based on the observations. Formally, a POMDP model can be represented by a tuple $(S, A, \Omega, T, R, O)$ [Cassandra, 1998], where $S$ represents a set of states, $A$ represents a set of actions, $\Omega$ represents a set of observations, $T$ is a state transition function, $R$ is a reward function, and $O$ is a set of conditional observation probabilities. As an extended version of the POMDP, the Interactive-POMDP (IPOMDP) presents a multiagent setting which enables an agent to model and predict behavior of other agents [Gmytrasiewicz and Doshi, 2005]. A Bayes-Adaptive POMDP (BA-POMDP) assumes that transition and observation probabilities are unknown or partially known [Ross et al., 2007].

### 2.4.3 Game-Theoretic Defensive Deception

In this section, we discuss game-theoretic DD techniques used in the literature. We use the following classification to discuss game-theoretic DD techniques for asset protection: honeypots, honeywebs, honeynets, obfuscation, deceptive signals, multiple DD techniques, fake objects, honey patches, deceptive network flow. Some studies used empirical game-theoretic experiment settings where players are human subjects. We also discussed them to show how game theory has been used to develop DD in empirical experimental settings with human-in-the-loop.

#### Honeypots

A honeypot has been studied as the most common DD strategy in the literature. A honeypot is mainly used in two forms: *low interaction honeypots* (LHs) and *high interaction honeypots* (HHs) [Han et al., 2018]. The differences between them are based on different deception detectability and deployment cost. HHs provide lower detectability by attackers than LHs while incurring higher deployment cost than LHs [Han et al., 2018].

Pawlick and Zhu [2015] employed a signaling game to develop a honeypot-based defense system where an attacker is able to detect honeypots. The authors investigated multiple models of signaling games with or without evidence when complete information is available or not. In the cheap-talk games with evidence (i.e., a signaling game with deception detection), extended from cheap-talk games (i.e., costless communication signaling game between a sender and receiver), the receiver (i.e., an attacker) is modeled to detect deception (i.e., a honeypot) with some probability. They found that the attacker's ability to detect deception does not necessarily lower down the defender's utility.

The signaling game with evidence has been also applied to model the honeypot selection or creation in other works [Pawlick et al., 2018; Píbil et al., 2012]. Pawlick et al. [2018] used a signaling game with evidence where the evidence is estimated based on a sender type and a transmitted message. The authors extended the signaling game with a detector with probabilistic evidence of deception. However, this signal is vulnerable to attackers which can analyze the signal, leading to detecting the deception and causing the evidence leakage. Píbil et al. [2012] introduced a game-theoretic high-interaction honeypot to waste attackers' resources and efforts. The authors designed a *honeypot selection game (HSG)* as a two-player zero-sum extensive-form game with imperfect and incomplete information. A honeypot is designed to mimic servers with high importance to provide cost-effective DD.

La et al. [2016] proposed a two-player attacker-defender deception-based Bayesian game in an IoT network. They used honeypots as a deceptive defense mechanism. They studied the Bayesian equilibrium in one shot and repeated games. Their findings showed the existence of a certain frequency of attacks beyond which both players will primarily take deception as their strategy. The authors used a honeypot as a deceptive strategy where a game is considered under incomplete information.

Çeker et al. [2016] proposed a deception-based defense mechanism to mitigate Denial-of-Service (DoS) attacks. The defender deploys honeypots to attract the attacker and retrieve information about the attacker's real intent. They used signaling games with perfect Bayesian equilibrium to model the interactions between the defender and attacker. Basak et al. [2019] used cyberdeception tools to identify an attacker type as early as possible to take better defensive strategies. The attacker's type is reflected in actions and goals when planning an attack campaign. The authors leveraged on a multistage Stackelberg Security game to model the interaction between the attacker and defender. In the game, the defender is a leader taking strategies considering the attacker's strategy. As a follower, the attacker selects its strategy after observing the leader's strategy. Through a game-theoretic approach, the defender selects deception actions (e.g., honeypots) to detect specific attackers as early as possible in an attack.

Mao et al. [2019] used honeypots as a defense strategy in a noncooperative Bayesian game with imperfect, incomplete information where an attacker is a leader and a defender is a follower. The authors considered a player's perception towards possible motivation, deceptions, and payoffs of the game. Kiekintveld et al. [2015] discussed several game models that address strategies to deploy honeypots, including a basic honeypot selection game. They first developed a honeypot selection game as a two-player zero-sum extensive-form game with imperfect and incomplete information in [Píbil et al., 2012]. Then they extended the game to allow additional probing actions by the attacker in which attack strategies are represented based on attack graphs [Kulkarni et al., July, 2021; Xi and Kamhoua, 2020]. The authors conducted experimental performance analysis to compare the performance of these model and discussed the advantages and disadvantages of the considered game-theoretic models in the DD research.

Durkota et al. [2015] leveraged a Stackelberg game where an attacker follows an attack graph and a defender can deploy a honeypot to deceive the attacker based on an efficient optimal strategy searching method using Markov Decision Processing (MDP) and sibling-class pruning. Game-theoretic cyberdeception techniques have been proposed using attack graphs [Kulkarni et al., 2020; Milani et al., 2020; Tsemogne et al., 2020]. Kulkarni et al. [2020] developed a zero-sum, hypergame of incomplete information for evaluating the effectiveness of the proposed honeypot allocation technique. Milani et al. [2020] developed a Stackelberg game to allocated defensive resources and manipulate a generated attack graph. Tsemogne et al. [2020] studied the malicious behavior through an epidemic model and solved a one-sided partially observable stochastic game for cyberdeception where a defender distributes a limited number of honeypots.

Wagener et al. [2009] addressed the issue of a honeypot being overly restrictive or overly tolerant by proposing a self-adaptive, game-theoretic honeypot. The authors modeled the game between an attacker and a defender by reusing the definitions proposed in [Greenwald, 2007]. They applied game theory to compute the optimal strategy profiles based on the computation of Nash Equilibrium. The game theory directs the configuration and a reciprocal action of the high-interaction honeypot. In their experiment, this technique produces an optimal strategy based on Nash equilibrium with rational attackers. Aggarwal et al. [2016, 2017] discussed a sequential, incomplete information game to model the attackers' decision-making in the presence of a honeypot and combined this model with instance-based learning (IBL). The authors introduced two timing-based deception by applying early or late deception in the rounds of games and investigated the effect of timing and the extent of deception. Their results showed that high amount of using deception and late timing of deception can effectively decrease attack actions on the network. However, using a different timing and amount of deception did not introduce any difference in attacking on a honeypot action.

Various honeypot allocation methods for deception over attack graphs have been studied based on game-theoretic approaches that considers uncertainty using partially observable MDP (POMDP) and stochastic games (POSG) [Anwar and Kamhoua, 2020; Kamhoua, 2018]. Anwar et al. [2019] developed a static game model to protect a connected graph of targeted nodes against an attacker through honeypot placements based on node features and significance. The game model is extended to study the game dynamics as a stochastic allocation game in [Anwar et al., 2020]. To resolve high complexity of attack graph-based honeypot allocation games, the authors proposed a heuristic approach.

Nan et al. [2019] and Nan et al. [2020a] provided Nash equilibrium-based solutions for a defender to intelligently select the nodes that should be used for performing a computation task while deceiving the attacker into expending resources for attacking fake nodes. Nan et al. [2019] investigated a two-player static game of complete information with mixed-strategy, where the system selects a node to place the deception source and the attacker selects a node to compromise. Nan et al. [2020a] constructed a static game with mixed-strategy, where both players select one strategy (target node) with probability distribution. The defender selects a node to install defense software while the attacker selects a target node to compromise and avoid being detected by the anti-malware software.

**Pros and Cons**: A honeypot is the most popular DD technology which has matured over decades. Since its aim is not to introduce any additional vulnerabilities, if an attacker is successfully lured by the honeypots, the honeypots can protect the existing system components (i.e., system assets) while collecting additional attack intelligence which can lead to improving intrusion detection with new attack signatures. However, maintaining honeypots incurs extra cost. In addition, there is still a potential performance degradation due to the existence of the honeypots, such as additional routing paths or resource consumption. However, the drawbacks of honeypots have been little investigated. In addition, as more intelligent, sophisticated attackers have been emerged recently, it becomes more challenging to develop realistic honeypots for effectively deceiving attackers in terms of both complexity and cost.

#### Honeywebs

El-Kosairy and Azer [2018] proposed a web server framework that provides a web application firewall with a honeyweb to detect malicious traffics and forward suspicious traffics to honeypot servers. This work formulated the interactions between the attacker (i.e., malicious traffics) and the defender (i.e., honey servers) as a game of imperfect information (i.e., players cannot not monitor each other's moves) but under complete information setting (i.e., players know their opponent's type, possible actions, and payoff function) considering simultaneous moves. Typically, this research combines deception technologies,

including honey token, honeypot, and decoy document. In the game definition, the main action of a defender is implementing a virtual machine (VM) to consume the attacker's effort and resources.
**Pros and Cons**: Honeywebs are rarely used particularly based on game-theoretic approaches. They are used to assist honeypots along with honey tokens or honey files. Developing fake webpages incurs less cost while its role is more like an assistant to support the honeypots. Hence, honeywebs may not be used without other defensive technologies.

### Honeynets

Garg and Grosu [2007] considered an attacker and a honeynet system (i.e., a defender) as players in a strategic noncooperative game. The framework is based on extensive games of imperfect information. In their game model, the defender makes deception about the placement of a honeypot while the attacker can probe the target host to identify their true roles with some probability. The attacker's utility considers the cost of both probing and compromising a host or honeypot. They studied the mixed strategy equilibrium solutions of these games and showed how they are used to determine the strategies of the honeynet system.

Dimitriadis [2007] proposed a honeynet architecture based on an attacker-defender game. This architecture is called 3GHNET (3G-based Honeynets) and aimed to enhance the security of the 3G core network by defending against DDoS and node compromise attack. The main components of this 3GHNET are two gateways with a set of strategies to control and capture the data flow between two nodes. In their emulation, each gateway and attacker are considered as individual players. The 3GHNET-G is a two-player, noncooperative and zero-sum game. The authors identified the optimal strategy based on Nash Equilibrium.
**Pros and Cons**: The honeynet is a system architecture designed to deal with inside attackers and allow a system manager to monitor/learn threats. The Honeynet Project [Spitzner, 2003] is a typical example of a honeynet consisting of multiple honeypots applied in practice. Since the honeynet works with multiple honeypots, how to optimally deploy and work together among them needs more investigation based on both the effectiveness of deception and additional deployment costs.

### Obfuscation

Shokri [2015] modeled a leader-follower game (i.e., Stackelberg game) between a designer of an obfuscation technique and a potential attacker. Authors designed adaptive mechanisms to defend against optimal inference attacks. They assumed that the users plan to protect sensitive information when sharing their data with untrustful entities. This enables the users to obfuscate the data with noises before sharing it. The attacker can observe valuable information of users and noises caused by obfuscation. They provided linear program solutions to search for optimal solutions that provably achieves a minimum utility loss under those privacy bounds.

Hespanha et al. [2000] proposed a DD framework based on noncooperative, zero-sum, stochastic games with partial information. They analyzed how a defender in a competitive game can manipulate information available to its opponents to maximize the effectiveness of the DD. They found that there exists an optimal amount of information to be presented to effectively deceive an attacker.
**Pros and Cons**: Compared to other DD techniques, such as honey-X techniques, the key benefit of data obfuscation is easy deployability with low cost. However, adding noise into normal information can

also confuse a defender or a legitimate user. On the other hand, most data obfuscation research mainly aims to develop a technique of how to hide real information rather than how to detect an attacker.

**Deceptive Signals**

In many game-theoretic approaches that consider DD as a defense strategy, deception is simply used as a signal to deceive an opponent player. Pawlick et al. [2015] modeled a game of three players, consisting of a cloud defender, an attacker, and a device, in a cloud-based system that can deal with advanced persistent threats (APTs). The authors designed the so-called *FlipIt* game to model the interactions between an attacker and a defender where signaling games are used to model the interactions between the device and the cloud which may be compromised with a certain probability. Xu and Zhuang [2016] designed a game between an attacker and a defender where the attacker can investigate the vulnerability of a target and the defender can apply defensive technologies to change the vulnerability of a certain device. This work mainly studied how the attackers' preparation for launching an attack affects the effectiveness of DD strategies. As the result, the authors applied the subgame perfect Nash Equilibrium (SPNE) to analyze the strategic interactions of the terrorist's costly learning and the defender's counter-learning.

Yin et al. [2013] leveraged a fake resource or converted a real resource to secrete recourse to mislead attackers. They applied a Stackelberg game to model the interactions between an attacker and a defender. The authors conducted the mathematical analysis of a game using both pure and mixed strategies played by the defender. They provided an optimal strategy for the defender and calculated the accuracy rate of identifying when the attacker can deploy unlimited surveillance before attacking. Horák et al. [2017] used one-sided POSG (partially observable stochastic games), in which a defender has perfect information while other players have imperfect information. In this work, an attacker and defender take sequences of actions to either deceive an opponent or attempt to obtain true actions taken by the opponent. This work assumed that a rational attacker has knowledge towards its opponent and its actions to take. However, the attacker has no knowledge of a network topology which introduces a hurdle to identify a target and whether the defender is aware of its presence in the network.

Ferguson-Walter et al. [2019b] proposed a hypergame-based DD scenario where players have imperfect and incomplete information. Based on the key concept of a hypergame, the authors modeled an individual player's own game where the game structure and payoffs may be manipulated by an opponent player. However, this work assumed an asymmetric view by the attacker and defender in that the attacker does not know the defenders taking DD strategies while the defender can know the attacker's true payoff and game structure.

Bilinski et al. [2019] introduced game-theoretic deception procedures based on a masking game in which a defender masks the true nature of a device. In this game, an attacker can ask the defender whether a specific device is real or fake at each round of the game. The defender needs to pay cost if it lies. After several rounds, the attacker chooses a target to attack. The authors investigated this scenario by conducting a mathematical analysis for nonadaptive and adaptive game models. They also designed a Stackelberg game model where the attacker is a leader and the defender is a follower. Through a Markov Decision Process (MDP) simulation, they investigated the potential behavior of an attacker at noncritical points.

**Pros and Cons**: This deceptive signal is an abstract way to apply a game theory concept to model a deception game between an attacker and a defender. Due to the nature of the abstracted game formulation, it has a high flexibility that can use various types of deception techniques. On the other hand, it is quite challenging to model a cybersecurity problem in a particular context as the proposed deception game framework does not use specific DD techniques.

### Multiple Defensive Deception Techniques

Some existing approaches leverage a set of cyberdeception technologies to model a set of strategies by a defender. Chiang et al. [2018] discussed DD to improve system security and dependability based on an SDN environment by utilizing a set of honey technologies. The authors discussed what are the critical requirements to realize effective DD and identified promising evaluation methods including metrics and evaluation testbeds. Chiang et al. [2018] constructed a dynamic game of complete information where the attacker does not know the strategies chosen by defender. Huang and Zhu [2019] discussed several DD techniques, such as honeypots, fake personal profiles, or multiple game models. They used a static Bayesian game to capture stealthy and deceptive characteristics of an attacker and advance this model by applying asymmetric information one-shot game. They demonstrated the performance of their proposed techniques under APT based on the Tennessee Eastman (TE) process as their case study.
**Pros and Cons**: Since multiple DDs are combined and used as a set of a defender's strategies, the defender can make more choices to defend against attackers based on different merits that can introduce to different types of attacks. In particular, to deal with APT attackers which can perform multi-staged attacks, using various types of DD can provide more relevant resources to deal with them. However, combing more than one deception techniques introduces additional deployment costs. In addition, it is not trivial to identify a optimal combination of multiple DD techniques.

### Fake Objects

Mohammadi et al. [2016] analyzed two-player signaling games with a defensive fake avatar for selecting an optimal strategy under various scenarios. A defender can use an avatar in the signaling game as a fake internal user to identify an external attacker by interacting with external users. This game considered the payoff of two players and derived an optimal threshold of raising alarms. In their experiment, the expected payoff is used a metric to guide actions of the avatar or the defender. Unlike other works using signaling games [Pawlick and Zhu, 2015; Pawlick et al., 2015], their signaling games put the defender as a second mover (i.e., a receiver) while an attacker is a first mover (i.e., a sender) where the games are played with incomplete information. The key idea of using the signaling games is creating uncertainty for the attacker because the defender uses a fake identity which can make the attacker doubtful about whether the receiver is a real user.

Casey et al. [2015] discussed interactions between an insider attacker and a defender of an organization. They considered a hostile agent that may obtain organization surface and harm the whole system. To mitigate this kind of threats, the authors proposed a *honey surface* to confuse the malicious agent by designing a basic compliance signaling game to model the interaction between agents and the organization. Casey et al. [2016] discussed the insider threat in an organization and applied a signaling game to model the interactions between agents with learning intelligence and a defender.

Thakoor et al. [2019] introduced a general-sum game, named *Cyber Camouflage Games* (CCGs), to model the interactions between a defender and an attacker performing reconnaissance attacks. The defender can mask the machine in the network with fake information, such as an operating system, to mitigate the effect of reconnaissance attacks. To identify an optimal strategy, they introduced the Fully

Polynomial Time Approximation Scheme given constraints and applied Mixed Integer Linear Program to search for an optimal strategy.

Zhu et al. [2013] simulated the infiltration of social honeybots-based defense into botnets of social networks. The authors proposed a framework, so-called SODEXO (SOcial network Deception and EXploitation) consisting of Honeybot Deployment (HD), Honeybot Exploitation (HE), and Protection and Alert System (PAS). HD is composed of a moderate number of honeybots. The HE considered the dynamics and utility optimization of honeybots and botmaster by a Stackelberg game model. The PAS chose an optimal deployment strategy based on the information gathered by honeybots. The results showed that a small number of honeybots can significantly decrease the infected population (i.e., a botnet) in a large social network.

**Pros and Cons**: Although using fake identities or avatars as a DD technique is less costly and relatively simple, it can introduce confusion to normal users and accordingly increase false alarms. However, little work has investigated the adverse effect of using fake identities, such as introducing confusion for legitimate users or producing extra vulnerabilities. In addition, the effect of intelligent attackers with high deception detectability has not been studied.

### Honey Patches

A honey patch mainly has two components: (1) A traditional patch to fix known software vulnerabilities; and (2) additional code to mislead an attacker to fake software vulnerabilities [Avery and Spafford, 2017]. In the literature, fake patches are also called *honey patches* [Araujo et al., 2014] or *ghost patches* [Avery and Spafford, 2017].

Araujo et al. [2014] coined the term *honey patches* that can function the same as a regular patch; but can efficiently redirect an attacker to a decoy, and allow the attacker to achieve a fake success. The game used in [Araujo et al., 2014] is a dynamic game of incomplete information where the payoff of attacker is unknown to defender. In addition, the authors provided a strategy allowing a web server administrator to transfer regular patches into honey patches. Enterprise scale systems are often exposed by the vulnerabilities due to the lack of boundary checking and subtle program misbehavior [Avery and Spafford, 2017]. A common way to eliminate the vulnerabilities is releasing security patches. However, as the security patches contain the location and types of vulnerability to be patched, an attacker could obtain a blueprint of system vulnerabilities by analyzing the security patches from the past. As a solution, Avery and Spafford [2017] provided a fake patch technique for misleading the attackers that try to analyze the historical patches. In particularly, this fake patch technique is designed to protect the patch that fixes input validation vulnerabilities. This fake patch can contain two parts. One is creating a decoy patch file to seduce the attacker and attract the attacker's attention away from the real patch file and the other is to alert defenders of potential intrusions or exfiltration attempts. This decoy file is generated by a modified technique [Bowen et al., 2009]. Another part is a bogus control flow for programs. This alternation misleads the reverse engineering, but does not change the output of the program. In their experiment, this technique was evaluated in [Cadar et al., 2008] where the program runtime and program analysis measure the effectiveness of this technique. The game and its components in [Avery and Spafford, 2017] were not rigorously presented since it mainly focused on the technicalities of developing fake honeypots. However, the deception and the evaluation of its reward represents a normal form one-shot game. Such games were clearly presented in [Avery and Wallrabenstein, 2018] where the authors evaluated the effectiveness of three deceptive patches (i.e., faux, obfuscated, and active response) and applying them into the proposed game-based module which provides the security guidelines. In addition, they found that many techniques are unable to meet the proposed security definition while emphasizing the importance of assessing deceptive patches based on a clear and meaningful security definition.

Cho et al. [2019] modeled a deception game based on hypergame theory in which an attacker and a defender have different perceptions of the game. This work examined how a player's (mis)perception can affect its decision making to choose strategies to take, which can affect the player's utility in the game. This work used Stochastic Perti Nets to build a probabilistic model of the hypergame where the defender uses fake patches as a deception strategy for misleading the attacker.

**Pros and Cons**: Honey patches are known as effective deception techniques to deceive attackers with low cost. However, as discussed in [Avery and Wallrabenstein, 2018], when some systematic and meaningful security assessments are conducted to validate the security of the honey patches, some honey patches fail to pass those security criteria, such as indistinguishablity between real and fake patches, vulnerabilities of fake patches, or cryptographic breakability for obfuscated messages [Avery and Wallrabenstein, 2018].

Table 2.3: Summary of Game-Theoretic-based Defensive Deception Techniques

| **Ref.** | **DD technique** | **Goal** | **Tactic** | **Expected effect** | **Main attacks** | **Game type** | **Domain** |
|---|---|---|---|---|---|---|---|
| [Cho et al., 2019] | Fake patch | Asset protection | Mimicking; False information; Lies | Luring; Confusing; Misleading | APT | Hypergame | No domain specified |
| [Ferguson-Walter et al., 2019b] | Deceptive signal | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | Hypergame | No domain specified |
| [Kiekintveld et al., 2015] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC/ probing | General-sum | CPS |
| [Mao et al., 2019] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | APT | Bayesian | SDN |
| [Pawlick and Zhu, 2015] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | Signaling | No domain specified |
| [Pawlick et al., 2015] | Deceptive signal | Asset protection | Mimicking; Masking; Lies; Decoying | Misleading; Luring; Confusing | APT | Three-player signaling | Cloud Web |
| [Clark et al., 2012] | Dummy packet generation | Asset protection | Mimicking; Masking | Misleading; Hiding; Confusing | Jamming | Stackelberg | Wireless |
| [Zhu et al., 2012] | Deceptive network flow | Asset protection | Mimicking; Masking | Misleading; Hiding; Confusing | Jamming | Multi-stage stochastic | No domain specified |
| [Mohammadi et al., 2016] | Fake avatar | Asset protection | Mimicking; Decoying | Asset protection | NC | Signaling | No domain specified |
| [Pawlick et al., 2018] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | Signaling | No domain specified |

**Table 2.3 – (Continued)**

| Ref. | DD technique | Goal | Tactic | Expected effect | Main attacks | Game type | Domain |
|---|---|---|---|---|---|---|---|
| [Píbil et al., 2012] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | General-sum | No domain specified |
| [La et al., 2016] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | Bayesian | IoT |
| [Çeker et al., 2016] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | DoS | Signaling | No domain specified |
| [Basak et al., 2019] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | APT | General-sum | No domain specified |
| [Xi and Kamhoua, 2020] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | APT | Static | IoT |
| [Durkota et al., 2015] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | Stackelberg | No domain specified |
| [Wagener et al., 2009] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | General-Sum | No domain specified |
| [Aggarwal et al., 2016] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC/ probing | Noncooperative sequential; incomplete information | No domain specified |
| [Aggarwal et al., 2017] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC/ probing | Sequential | No domain specified |
| [Anwar and Kamhoua, 2020] | Deceptive signal | Asset protection | Mimicking; Masking; Decoying | Misleading; Luring; Confusing | NC/ probing | Stochastic with POMDP | IoT |
| [Anwar et al., 2019] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | Stochastic | IoT |
| [Anwar et al., 2020] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | Stochastic | IoT |
| [Nan et al., 2019] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | NC | Static | IoT |
| [El-Kosairy and Azer, 2018] | Honeywebs | Asset protection | Decoying; Mimicking | Blending; Misleading; Luring | Web attack | Complete, imperfect game | Cloud Web |
| [Garg and Grosu, 2007] | Honeynet | Asset protection; Attack detection | Decoying; Mimicking | Asset protection; Attack detection | NC | Zero-sum static | No domain specified |
| [Dimitriadis, 2007] | Honeypot | Asset protection; Attack detection | Decoying; Mimicking | Blending; Misleading; Luring | DDoS | Zero-sum static | Wireless |

**Table 2.3 – (Continued)**

| Ref. | DD technique | Goal | Tactic | Expected effect | Main attacks | Game type | Domain |
|---|---|---|---|---|---|---|---|
| [Hespanha et al., 2000] | Obfuscation | Asset protection | Masking | Misleading; Confusing | NC | Noncooperative stochastic | No domain specified |
| [Yin et al., 2013] | Deceptive signal | Asset protection | Masking; Camouflaging; Mimicking | Hiding; Blending, Hiding; Misleading | Reconnaissance | Stackelberg | No domain specified |
| [Horák et al., 2017] | Deceptive signal | Asset protection; Attack detection | Misleading; Hiding; Mimicking; Decoying | Misleading; Hiding; Luring; Confusing; Blending | NC | Stochastic | No domain specified |
| [Bilinski et al., 2019] | Deceptive signal | Asset protection; Attack detection | Misleading; Hiding; Mimicking; Decoying | Asset protection; Attack detection | NC | Stackelberg | No domain specified |
| [Chiang et al., 2018] | Honey-X technologies | Asset protection; Attack detection | Misleading; Hiding; Mimicking; Decoying | Misleading; Hiding; Luring; Confusing; Blending | APT | General-sum | SDN |
| [Huang and Zhu, 2019] | Deceptive signal | Asset protection; Attack detection | Misleading; Hiding; Mimicking; Decoying | Misleading; Hiding; Luring; Confusing; Blending | APT | Dynamic Bayesian | No domain specified |
| [Casey et al., 2015] | Honey surface | Asset protection | Mimicking; Decoying | Hiding; Misleading | Insider | Signaling | No domain specified |
| [Casey et al., 2016] | Honey surface | Asset protection | Mimicking; Decoying | Hiding; Misleading | Insider | Signaling | No domain specified |
| [Thakoor et al., 2019] | Fake objects | Asset protection | Mimicking; Decoying | Hiding; Misleading | Recon. | General-sum | No domain specified |
| [Zhu et al., 2013] | Social honeypots | Asset protection; Attack detection | Decoying; Mimicking | Blending; Misleading; Luring | Social bots | Stackelberg | No domain specified |
| [Sayin and Başar, 2019] | Deceptive network flow | Asset protection | Mimicking; Masking | Misleading; Hiding; Confusing | Recon./ probing | Non-zero-sum Stackelberg | CPS |
| [Al Amin et al., 2019a] | Honeypot | Asset protection; Attack detection | Decoying; Mimicking | Blending; Misleading; Luring | APT | POMDP | SDN |
| [Al Amin et al., 2020] | Honeypot | Asset protection; Attack detection | Decoying; Mimicking | Blending; Misleading; Luring | APT | POMDP | SDN |
| [Nan et al., 2020b] | Deception signal | Asset protection; Attack detection | Decoying; Mimicking | Blending; Misleading; Luring | NC | Stackelberg | Wireless |
| [Wang et al., 2020] | Honeypot | Asset protection; Attack detection | Decoying; Mimicking | Blending; Misleading; Luring | APT | Q-learning-based | No domain specified |

**Table 2.3 – (Continued)**

| Ref. | DD technique | Goal | Tactic | Expected effect | Main attacks | Game type | Domain |
|---|---|---|---|---|---|---|---|
| [Al Amin et al., 2019b] | Fake nodes | Asset protection; Attack detection | Decoying | Hiding; Misleading | APT | POMDP | No domain specified |
| [Rahman et al., 2013] | Fake objects | Asset protection | Mimicking; Decoying | Hiding; Misleading | Reconnaissance | Signaling | No domain specified |
| [Shi et al., 2020] | Feature deception | Asset protection | Lies | Misleading | Reconnaissance | RL-based feature deception | No domain specified |

### Deceptive Network Flow

Clark et al. [2012] proposed the deceptive network flow, representing the flow of randomly generated dummy packets. They assumed that both of real and deceptive packets are encrypted. Hence, an attacker cannot distinguish between them and may spend limited resources on targeting a false flow. This work used this deceptive flow to lure the attacker and waste its resource to assist those real packets to protect the network from jamming attacks. The main challenges of developing defensive network flow are: (1) applying deceptive packets may increase the risk of congestion and incur extra delay for the delivery of real packets; and (2) the source node has a limited capacity to generate and transfer packets, requiring a balance between real and fake flows. To mitigate this adverse effect, the authors designed a two-stage game model to obtain deception strategies at pure-strategy Stackelberg equilibrium. They considered two types of source nodes: (1) *selfish* nodes aiming to maximize its own utility; and (2) *altruistic* nodes considering the congestion of other sources when choosing a flow rate. Their results proved that altruistic node behavior improves the overall utility of the sources. Similarly, Zhu et al. [2012] considered a single source selecting routing paths for real and deceptive flows. Before sending a deceptive flow, the source node is allowed to choose the rate of deceptive and real flow as well as the path of the deceptive flow. This work introduced the solution concepts, such as the path Stackelberg equilibrium, the rate Stackelberg equilibrium, and their mixed strategy counterparts of the game. Their results proved that there exist such equilibria. Anjum et al. [2020] designed a deceptive network flow system, called *Snaz*, to mislead the attacker performing reconnaissance attacks. They model the interaction between the attacker and the defender with a two players non-zero-sum Stackelberg game. Sayin and Başar [2019] proposed a deceptive signaling game framework to deal with APT attacks in cyber-physical systems. Under the attacks aiming to obtain system intelligence via scanning or reconnaissance attacks, this work crafted bait information to lure the attackers. This work leveraged the concept of game-theoretic hierarchical equilibrium and solved a semi-definite programming problem where the defender does not have perfect information by considering partial or noisy observations or uncertainty towards the attacker's goal.

**Pros and Cons**: Deceptive network flow is a new DD approach to effectively mitigate some malicious actions which cannot be defended by traditional DD technologies, such as malicious network flow scanning and fingerprint. However, it is highly challenging to balance fake and real network flows that minimizes any additional network performance degradation.

In Table 2.3, we summarized the five key aspects (i.e., deception techniques, categories, expected effects, attacks considered, and application domain) of each game-theoretic DD technique surveyed in this work.

### 2.4.4 Empirical Game Experiments Using Human Subjects

As we observed in Section 2.4.3, most deception game-theoretic works have been studied based on simulation testbeds. However, cognitive scientists (specializing in decision making) have conducted empirical experiments using human subjects which are assigned as attackers or defenders to consider a deception game. Cranford et al. [2020, 2019] conducted an empirical experiment to realize a signaling game between an attacker and a defender where the defender uses deceptive signals. In this experiment, the players are humans with limited cognition, reflecting bounded rationality in game theory. To measure the effectiveness of DD techniques, Ferguson-Walter et al. [2018] and Ferguson-Walter [2020] also conducted an empirical experiment with 130 red team members as participants in a network penetration study at different deception scenarios. For instance, different types of decoy devices are described explicitly for the existence of deceptive defensive techniques to the participant. The results showed that when the participants are aware of the deception, they took much more time before taking any action. This shows that deception made them slower to move and attack. Aggarwal et al. [2019] developed a simulation tool, called *HackIt*, to study attacks in a network reconnaissance stage where participants studied the effect of introducing deception at different timing intervals. Their results showed that the attacker performed attacks on the honeypots more often than real machines.

Pure game theorists may argue that this type of empirical game experiment is not a game-theoretic deception game. However, we discuss these empirical deception game studies in this paper to understand the role of human subjects-based empirical game experiments. Such games can partly follow the structure of traditional game-theoretic models. For instance, it considers two or more players, with bounded rationality, opponent strategies, and corresponding utilities.

## 2.5 Machine-Learning-Based Defensive Deception (MLDD)

ML-based applications become more popular than ever in various domains, including cybersecurity. ML techniques have been extensively adopted for automating attacks and learning system behaviors in the cyberdeception domain [Al-Shaer et al., 2019]. In this section, we first discuss the key steps to implement ML-based approaches and conduct an extensive survey on ML-based defensive deception (DD) techniques in the literature.

### 2.5.1 Key Steps of Implementing ML-Based Defensive Deception

Like other ML-based applications, ML-based deception techniques also commonly use the following steps to detect malicious activities:

- *Dataset Generation*: ML-based approaches have been mainly used in detecting attackers or identifying malicious activities based on the information collected in honeynets or honeypots [Song et al., 2011].
- *Dataset Collection*: The success of ML techniques hugely depends on whether reliable datasets are available or not. However, obtaining reliable cybersecurity datasets is not trivial based on the following reasons: (i) such datasets are often confidential due to security reasons of a company or an organization owning the datasets; and (ii) there are inherent unknown attacks which make it difficult to obtain accurate and complete annotated datasets. Cybersecurity related datasets have

been more available as more security applications are using ML-based approaches [Snijders et al., 2012].

- *Pre-Processing*: Datasets should be carefully pre-processed ahead of the learning step to eliminate unwanted noises from the dataset. This includes data cleaning, editing, and reduction [Han et al., 2011].
- *Feature Extraction*: It is challenging to analyze complex data because of the potentially numerous variables involved. The goal of feature extraction is to construct a combination of the relevant variables to describe the data with sufficient accuracy.
- *Training*: Training dataset is a dataset of examples used during the learning process and is used to learn the parameters of the detector [Han et al., 2011; Ron and Foster, 1998].
- *Testing*: A testing dataset is a set of examples used to evaluate the performance (i.e, detection accuracy) of a trained detector [Ron and Foster, 1998].
- *Evaluation*: Independent and new data are used to evaluate the performance of a detector to avoid overfitting problems in the data [Ron and Foster, 1998].

### 2.5.2 Common ML Techniques Used for Defensive Deception

We discuss popular ML techniques used to develop various DD techniques in the literature. For the easy understanding of readers who may not have enough background on ML, we brief the overview of the following ML techniques:

- *Support Vector Machine (SVM)*: SVMs can deal with a set of supervised learning models and be applied to solve classification and regression problems [Cortes and Vapnik, 1995]. SVM uses the hypothesis space of linear functions in a high dimensional feature space to analyze data. SVM is popular due to its simple training process. However, SVM is not efficient for large or highly noisy datasets.
- *$K$-Means*: This uses an iterative refinement to partition $n$ observations into $k$ clusters and guarantee that each observation belongs to a cluster with the nearest mean [Wikipedia contributors, 2024].
- *Expectation Maximization (EM)*: This statistically searches for maximum likelihood parameters with two major steps: (1) an expectation step, which creates a function for the expectation of the likelihood evaluated using the current estimate for the parameters; and (2) a maximization step, which computes a new estimate of the parameters [Moon, 1996].
- *Hierarchical Grouping*: This is a well-established classification method for cluster analysis which seeks to build a hierarchy of clusters [Ward Jr, 1963]. For example, plants and animals may be grouped into a smaller number of mutually exclusive classes with respect to their genetic characteristics.
- *Bayesian Network (BayesNet)*: This is a classifier that learns the conditional probability of each attribute from training data while assuming strong independence [Friedman et al., 1997]. That is, the underlying probability model would be an independent feature model such that the presence

of a particular feature of a class is unrelated to the presence of any other features. For example, BayesNets could represent the probabilistic relationships between diseases and symptoms.

- *Decision Tree (DT)*: DT is widely used for multi-stage and sequential decision making. The key idea of DT lies in breaking up a complex decision into a union of several simpler decisions, leading to the intended desired solution [Safavian and Landgrebe, 1991].
- *C4.5 Algorithm*: This generates a decision tree based on the concept of information entropy in which the DT generated by C4.5 is widely used for data classification [Quinlan, 2014].
- *Naïve-Bayes Algorithm*: This is a widely applied classification algorithm based on Bayesian Theory and statistical analysis [Domingos and Pazzani, 1997]. This algorithm uses the prior and posterior probabilities of the training data to calculate the probability of type (posterior probability) of a specific sample.
- *Deep Neural Networks*: DNN is a neural network with at least one hidden layer and has been used for deep learning. For each node (neuron), its output is a nonlinear function of a weighted sum of its inputs. The DNN model is trained by back propagation, which changes the weight of each node. DNNs have been used for natural language processing, image recognition, and disease diagnosis.

### 2.5.3 ML-Based Defensive Deception

This section discusses ML-based DD techniques discussed in the literature. We discuss the following DD techniques along with what ML techniques are used to develop them: Social honeypots, honey files, honeypots or decoy systems, bait-based deception, fake services, obfuscation, and decoy data.

#### Social Honeypots

In online social networks (OSNs), the so-called *social honeypots*, as good bots in contrast to bad bots, have been studied by creating social network avatars.

Lee et al. [2010] proposed an ML-based mechanism to discover social spammers. The authors used social honeypots as a DD technique to monitor spammer activities. The data from spam and legitimate profiles are used to improve an ML-based classifier. Thus, the system can classify spam profiles automatically. The paper compared true positive rate (TPR) and false negative rate (FNR) of various ML-based classifiers to show the effectiveness of the proposed mechanism. Lee et al. [2011] developed social honeypots to detect content polluters in Twitter. Sixty social honeypot accounts are created to follow other social honeypot accounts and post four types of tweets to each other. This work categorized the collected user features into nine classes based on the Expectation-Maximization (EM) algorithm. They classified the content polluters based on Random Forest and enhanced the results by using standard boosting and bagging and different feature group combinations.

El Idrissi Younes et al. [2016] developed social honeypots to discover malicious profiles. The authors employed feature-based strategies as well as honeypot feature-based strategies to collect data of the malicious profiles and analyzed the collected data based on a suite of ML algorithms provided by the Weka ML toolkit (e.g., logistic regression, Bayes classifier, support vector machine (SVM), $K$-Means, Expectation Maximization, or hierarchical grouping). Zhu [2015] introduced the concept of *active honeypots* as active Twitter accounts that can capture more than 10 new spammers every day. They identified

1,814 accounts from Twitter and examined the key features of active honeypots. In addition, using a suite of ML algorithms, such as SVM, logistic regression, J48 Tree, Bagging, and AdaBoost M1, they examined the impact of unbalanced datasets on the detection accuracy of the different ML algorithms.

Badri Satya et al. [2016] also used social honeypot pages to collect fake likers on Facebook where fake likes are provided by paid workers. The authors obtained four types of user profiles and behavioral features as the distinctive patterns of fake likers. This work evaluated the robustness of their ML-based detection algorithms based on synthetic datasets which have been modified to reflect individual and coordinated attack models. Yang et al. [2014] proposed a passive social honeypot to collect a spammer's preferences by considering various behaviors of the social honeypot. The considered tweet design features include tweet frequency, tweet keywords, and tweet topics as well as the behaviors of famous users' accounts and application installation. The authors conducted in-depth analysis on what types of social honeypot features can introduce a higher rate of collecting social adversaries and enhanced the social honeypots based on the result. The improved honeypot has shown 26 times faster than a normal social honeypot in collecting social adversaries based on the evaluation using a random forest classifier.

**Pros and Cons**: The works discussed in [El Idrissi Younes et al., 2016; Lee et al., 2010, 2011; Yang et al., 2014] mainly relied on luring deception techniques to attract spammers in OSNs and collect more attack intelligence particularly in terms of novel behavioral characteristics of spammers. Some limitations of the existing social honeypots include: (i) the current efforts are mainly to detect spammers, not other social network attacks (e.g., cybergroomers, human trafficking, cyberstalking, or cyberbullying); (ii) most features are based on user behaviors, but not on network topological features as an attacker's characteristics; (iii) most detectors are applied on static datasets to evaluate the detection of spammer. This implies that there is no theoretical simulation framework that considers dynamic interactions between an attacker and a defender; and (iv) there is a lack of studies investigating the effectiveness of social honeypots in terms of how quickly more attack intelligence can be collected and what types of attack intelligence are collected.

### Honey Profiles

Stringhini et al. [2010] analyzed 900 honey profiles for detecting spammers in three online social networks, including MySpace, Facebook, and Twitter. The authors collected users' activity data for a year where the user datasets include both spammers' and legitimate users' profiles and the honey profiles are spread in different geographic networks. Further, this work captured both spam profiles and spam campaigns based on the shared URL using machine learning techniques (e.g., SVM).

**Pros and Cons**: Honey profiles allow identifying a large set of attackers, such as a large-scale spam bot farms, leading to detecting large-scale and coordinated campaigns. However, the existing spam detectors using honey profiles is not generic and needs to be tailored depending on a different social media platform.

### Honeypots or Decoy Systems

Nanda et al. [2016] used a suite of machine learning algorithms to train historical network attack data in software-defined networks for the development of quality honeypots. This ML-based method is to identify potential malicious connections and attack destinations. The authors employed four well-known ML algorithms, including C4.5, Bayesian Network (BayesNet), Decision Table (DT), and Naïve-Bayes to predict potential victim hosts based on the historical data. To prevent call fraud and identity theft attacks, Krueger et al. [2012] proposed a method called *PRISMA* (PRotocol Inspection and State Machine Analysis). This method is developed to infer a functional state machine and a message format of a protocol from only network traffic. The authors experimented to validate the performance of the PRISMA based on three real-world network traces datasets, including 10K to 2 million messages in both binary and textual protocols. Their experiments proved that the PRISMA provides the capability to simulate complete and correct sessions with the learned models and execute different states for malware analysis.

Hofer et al. [2019] discussed the attributes of cyber-physical decoys to design a system with the attributes to develop deception decoys. They integrated the deception decoys into real systems to make them harder to detect and more appealing as targets. To increase the fidelity of the added decoy devices to the physical system, three attributes, a protocol, variables, and a logic of the deployed decoy devices, are maintained. The authors trained a recurrent neural network (RNN) using a dataset collected for a year to learn such system attributes.

Some DD games have been studied by considering game-theoretic reinforcement learning (RL) where RL is considered in formulating players (i.e., attackers or defenders)'s utilities. That is, in the RL-based game formulation, players use an RL to identify an optimal strategy where the RL's reward function considers gain and loss based on the player's belief towards an opponent's move. RL-based deception games are studied in [Al Amin et al., 2019b; Wang et al., 2020]. Al Amin et al. [2019b] proposed an online deception approach that designs and places network decoys considering scenarios where a defender's action influences an attacker to dynamically change its strategies and tactics while maintaining the trade-off between availability and security. The defender maintains a belief consisting of a security state while the resultant actions are modeled as Partially Observable Markov Decision Process (POMDP). The defender uses an online deception algorithm to select actions based on the observed attacker behavior using a POMDP model. This embedded reinforcement learning (RL)-based model assumes that the defender belief about the attacker's progress is observed through an network-based intrusion detection system (NIDS). The defender hence takes actions that attract the attacker toward decoy nodes. Wang et al. [2020] identified an optimal deployment strategy for deception resource, such as honeypots. The authors developed a Q-learning algorithm for an intelligent deployment policy to dynamically place deception resources as the network security state changes. They considered an attacker-defender game by analyzing an attacker's strategy under uncertainty and a defender's strategies with several deployment location policies. They used RL with a Q-learning training algorithm that identifies an optimal deployment policy for deception resources.

**Pros and Cons**: In honeypots, ML is mainly used to detect attacks based on the attack intelligence gathered in honeypots. Hence, this research direction is like developing an intrusion detection mechanism, rather than creating better quality DD techniques. Various ML-based techniques can be highly leveraged to create real-like honeypots and its effectiveness can be measured based on the volume of attackers caught in honeypots and the diversity of attacker types.

### Bait-Based Deception

Ben Salem and Stolfo [2012] developed a bait-based deception to improve the detection of impersonation attacks by using the features combining a user behavior profiling technique with a baiting deception technique. This work used highly crafted and well-placed decoy documents to bait attackers and deployed a detector that uses SVM to model the user's search behavior. Whitham [2016] used bait-based deception

Table 2.4: Summary of Machine Learning-based Defensive Deception Techniques

| **Ref.** | **DD technique** | **Goal** | **Tactic** | **Expected effect** | **Main attacks** | **ML technique** | **Domain** |
|---|---|---|---|---|---|---|---|
| [Nanda et al., 2016] | Honeypot | Attack detection | Mimicking | Attack detection | Anomaly network traffic | C4.5, Bayesian Network, Decision Table, and Naïve-Bayes | No domain specified |
| [Badri Satya et al., 2016] | Social honeypot | Attack detection | Mimicking | Luring | Fake liker (crowdturfing) | Supervised ML | OSN |
| [El Idrissi Younes et al., 2016] | Social honeypot | Attacker identification | Mimicking | Luring | Malicious profile | ML toolkit | No domain specified |
| [Krueger et al., 2012] | Honeypot | Asset protection; Attack detection | Mimicking; Decoying | Luring; Misleading | Call fraud; Identity theft | ML toolkit | No domain specified |
| [Lee et al., 2010] | Social honeypot | Identifying spammers | Mimicking; Decoying | Luring | Social spammers | ML toolkit | OSN |
| [Lee et al., 2011] | Social honeypot | Identifying spammers | Mimicking; Decoying | Luring | Social spammers | ML toolkit | OSN |
| [Hofer et al., 2019] | Decoy | Asset protection; Attack detection | Decoying; Mimicking | Blending; Misleading; Luring | APT | RNN | CPS |
| [Ben Salem and Stolfo, 2012] | Bait-based Deception | Asset protection; Attack detection | Mimicking; Lies | Luring; Confusing | Insider | SVM | No domain specified |
| [Whitham, 2016] | Bait-based Deception | Attack protection | Mimicking; Lies | Luring; Confusing | Zero-day | NLP | No domain specified |
| [Stringhini et al., 2010] | Honey profile | Asset protection | Mimicking; Decoying | Hiding; Misleading | Spamming | SVM | OSN |
| [Whitham, 2017] | Bait-based Deception | Attack detection | Mimicking | Luring | Zero-day | NLP | No domain specified |
| [Abay et al., 2019] | Decoy data | Attack detection | Lies | Misleading | Data leakout | DL | No domain specified |

to study and evaluate honey files in military networks. The main challenge in military networks is the need to employ third-party resources such as cloud services. Third-party servers are directly connected to the Internet and may store sensitive material and information which represents a security challenge. The author proposed three new automated honey file designs that minimize the replication of classified material, yet remain enticing to malicious software or user-driven searches. Moreover, this work presented an NLP-based content generation design for honey files.

**Pros and Cons**: Although bait-based deception can contribute to increasing intrusion detection by collecting more intelligence during the time of the deception, there is no performance guarantee based on bait-based deception because the attackers may not be interested in the bait. In addition, to attract attackers more effectively, if more important information is used as a bait, the bait itself introduces risk when intelligent attackers can derive clues of system vulnerabilities based on the baits they have examined. Thus, mixing real with fake information to avoid too high a risk can provide an alternative, such as semi-bait-based deception.

### Fake Services

To increase the effectiveness and efficiency of DD techniques, real systems have been used to deceive attackers. ML-based fake system modification is used as a DD mechanism [Al Amin et al., 2019a, 2020] using POMDP to consider uncertainty in learning attack behavior. The authors designed a deception server that can modify IP addresses of fake network via DHCP server, DNS server, and forwarding ARP requests by appropriate flow rules to the deception server. In addition, the deception server is designed to modify the requests and sent it back to the requesting host. Shi et al. [2020] considered the changes of system configurations to deceive an attacker after learning its preferences and behavior from attack data.

**Pros and Cons**: Proving fake services to attackers can effectively increase attack cost or complexity which can delay launching attacks or make the escalation of attack fail. However, this technique can be deployed based on the assumption of accurate detection of intrusions. As the current intrusion detection mechanisms suffer from high false positives in practice, if a normal user is treated as an intrusion and fake services are provided to the user, it hinders normal availability of service provisions to legitimate users.

#### Decoy Data

Machine and deep learning (DL) algorithms have played a key role in developing decoy data, creating decoy objects or honey information. Abay et al. [2019] took a decoy data generation approach to fool attackers without degrading system performance by leveraging DL. To be specific, their approach employs unsupervised DL to form a generative neural network that samples HoneyData.

**Pros and Cons**: Since the defender launches false information injection (or data disruption) attack while an attacker stays in a system, this technique will 'scare off' attackers performing highly detectable attacks. However, it can also be disruptive to ordinary users if fake or decoy information is allowed to remain in the system.

As we observed in the extensive discussions of ML-based DD techniques above, ML-based DD has been addressed mainly with the following aims: (1) Protecting system components by hiding real components or creating fake things (i.e., honey-X) that look like real; and (2) Detecting attacks (or identifying attackers) by creating fake objects (e.g., honeypots) and accordingly attract attackers to collect attack intelligence. Therefore, the questions of what ML techniques are used and how are closely related to the quality of DD techniques whose success is closely related to how well they can deceive attackers, representing the quality of deception.

In Table 2.4, we summarized the five key aspects (i.e., deception techniques, categories, expected effects, attacks considered, and application domain) of each ML-based DD technique surveyed in this work.

## 2.6 Attacks Counteracted by Defensive Deception

A variety of attacks have been countered by existing defensive deception (DD) techniques as follows:

- *Scanning (or reconnaissance) attacks* [Al Amin et al., 2019a; Anjum et al., 2020; Avery and Wallrabenstein, 2018; Bilinski et al., 2019; Chiang et al., 2018; Mao et al., 2019; Sayin and Başar, 2019; Xu and Zhuang, 2016]: An outside attacker may aim to obtain a target system's information and identify vulnerable system component to penetrate the target system. This attack is often considered as part of the cyber kill chain in APT attacks. In the literature, the scanning attacks are categorized into two types as follows:
  - *Passive monitoring attacks* [Anjum et al., 2020; Bilinski et al., 2019; Xu and Zhuang, 2016]: Leveraging compromised nodes (e.g., routers or switches), attackers can scan traffics passing

through the nodes and analyze packets to gather information about hosts. Passive monitoring can make the attackers achieve active nodes' discovery, OS, roles, up-time, services, supporting protocols, and IP network configuration [Montigny-Leboeuf and Massicotte, 2004]. However, passive monitoring cannot identify services that do not run on well-known ports or are misdirected without protocol-specific decoders [Bartlett et al., 2007].

  – *Active probing attacks* [Al Amin et al., 2019a; Avery and Wallrabenstein, 2018; Chiang et al., 2018; Mao et al., 2019; Sayin and Başar, 2019]: Through sending probing packets to a potential target, the attacker can obtain the machine's information, including OS, opened ports, and installed application version. Although active probing can allow the attackers to collect more system configurations, it is relatively easy to be detected by traditional defensive technologies, such as IDS [Bartlett et al., 2007].

- *Network fingerprinting* [Rahman et al., 2013]: To analyze a target network and identify ideal target nodes, an attacker uses fingerprint tools to identify specific features of a network or a device, such as a network protocol or OS type. Common fingerprint tools include Nmap [Lyon, 2024], P0f [Zalewski, 2014], and Xprobe [Arkin and Yarochkin, 2002]. Some deception techniques, such as deceptive network flows and honeypots, focuses on addressing this kind of attacks.

- *(Distributed) Denial-of-Service* (DoS) [Çeker et al., 2016; Dimitriadis, 2007]: DoS attacks are mainly to make legitimate users be denied from proper services. This attack can overwhelm or flood network flow to a targeted machine to disrupt normal functions or communications. A common DoS attack is ping flooding by leveraging ICMP (ping) packets to overwhelm a target [Peng et al., 2007]. In distributed environments, distributed DoS (DDoS) flooding attacks can be performed with the intent to block legitimate users from accessing network resource(s). With flooding packets to a target, DDoS attackers can exhaust network bandwidth or other resources, such as a server's CPU, memory, or I/O bandwidth. Unlike DoS attacks, DDoS attackers can remotely control several devices as its botnet can continually send a large amount of traffics to a target to block the normal functions and services [Zargar et al., 2013].

- *Malware* [Krueger et al., 2012]: Malware refers to any software aiming to introduce some damage to a system component, such as server, client, or network. Honeypot-based deception approaches have been popularly used for malware analysis [Krueger et al., 2012].

- *Privacy attacks* [Shokri, 2015]: An attacker can identify private information while training and processing data to develop DD techniques. Obfuscation is often used to defend against privacy attacks.

- *Advanced persistent threats* (APT) [Cho et al., 2019; Hofer et al., 2019; Huang and Zhu, 2019; La et al., 2016; Pawlick et al., 2015, 2018; Shi et al., 2020]: An APT attack is the most well-known sophisticated attack performing different types of attacks in the stages of the cyber kill chain (CKC) [Okhravi et al., 2013]. In particular, game-theoretic DD research has considered the APT attack but mostly focused on reconnaissance attacks.

- *Spamming* [Lee et al., 2010, 2011; Stringhini et al., 2010; Yang et al., 2014; Zhu, 2015]: Online users may receive unsolicited messages (spam), ranging from advertising to phishing messages [Rathore et al., 2017].

- *Malicious or fake profiles* [El Idrissi Younes et al., 2016] (a.k.a. Sybil attacks): OSN attackers can create numerous fake identities to achieve their own selfish goal based on others' personal information, such as e-mail, physical addresses, date of birth, employment date, or photos.

- *Fake likers by crowdturfing* [Badri Satya et al., 2016]: Human attackers, paid by malicious employers, can disseminate false information to achieve the malicious employers' purposes via crowdsourcing systems [Wang et al., 2012]. This is called *crowdturfing* and mostly aimed to spread fake information to mislead people's beliefs.

- *Loss of confidentiality, integrity, and availability (CIA)* [Casey et al., 2016; Cranford et al., 2020, 2019]: An insider attacker, called a *traitor* [Salem et al., 2008], can leak out confidential information to the outside as a legitimate user. Further, by using its legitimate status with the authorization of a target device, it can perform attacks (e.g., illegal access, modification of confidential information) that can lead to loss of integrity and availability.

- *Impersonation attacks* [Ben Salem and Stolfo, 2012]: An inside attacker can masquerade a legitimate user's identity to access resources in a target system component [Salem et al., 2008].

- *Jamming attack* [Clark et al., 2012; Nan et al., 2020b; Zhu et al., 2012]: A jamming attack can be seen as a subset of DoS attacks. The difference is that jamming attacks mainly focus on incurring traffics in wireless networks while the DoS attacks can be applicable in all networks. The jamming attack is relatively simple to archive. By keeping flooding packets, a jamming attacker can effectively block the communication on a wireless channel, disrupt the normal operation, cause performance issues, and even damage the control system [Grover et al., 2014].

- *Node compromise (NC)* [Aggarwal et al., 2016, 2017; Anwar et al., 2020; Anwar and Kamhoua, 2020; Anwar et al., 2019; Basak et al., 2019; Bilinski et al., 2019; Durkota et al., 2015; Ferguson-Walter et al., 2019b; Garg and Grosu, 2007; Horák et al., 2017; Kiekintveld et al., 2015; La et al., 2016; Nan et al., 2019, 2020a; Pawlick and Zhu, 2015; Pawlick et al., 2018; Píbil et al., 2012; Wagener et al., 2009; Xi and Kamhoua, 2020]: Some research does not specify the details of attack process. The authors only use "device compromising" to represent an attack. Some research discusses that an attacker can probe a target before attacking [Aggarwal et al., 2016, 2017; Anwar et al., 2020; Anwar and Kamhoua, 2020; Anwar et al., 2019; Ferguson-Walter et al., 2019b; Garg and Grosu, 2007; Kiekintveld et al., 2015; La et al., 2016; Nan et al., 2019, 2020a; Píbil et al., 2012; Wagener et al., 2009; Xi and Kamhoua, 2020] while others only discuss the attacking actions [Basak et al., 2019; Bilinski et al., 2019; Durkota et al., 2015; Horák et al., 2017; Pawlick and Zhu, 2015; Pawlick et al., 2018].

- *Web attack* [El-Kosairy and Azer, 2018]: An attacker can leverage existing vulnerabilities in web applications to gather sensitive data and gain unauthorized access to the web servers.

- *Zero-day attack* [Whitham, 2016, 2017]: An attacker can exploit the vulnerability period where an unknown vulnerability is not mitigated by a defense system. For example, before a vulnerability is patched by the system, the attacker can exploit the vulnerability to launch the attack, which is called zero-day attack. Often, DD along with moving target defense (e.g., network address or topology shuffling [Djamaluddin et al., 2018]) can mitigate this attack effectively.

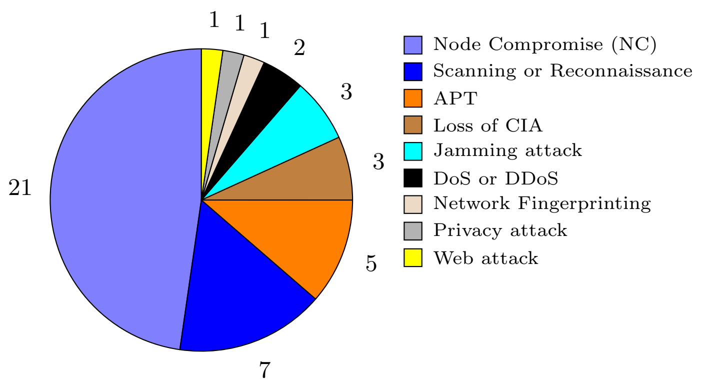


Figure 2.2: Types and frequency of attacks considered in the existing game-theoretic defensive deception approaches.

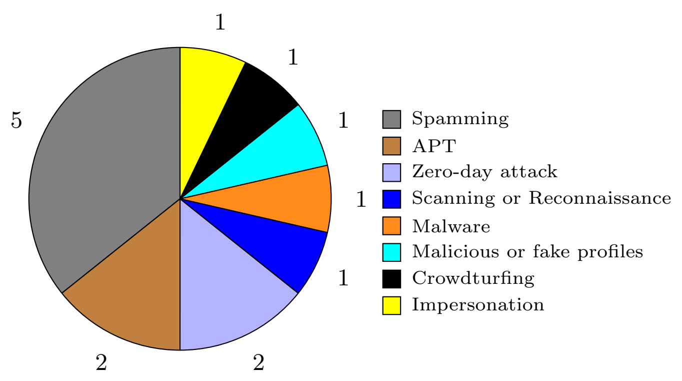


Figure 2.3: Types and frequency of attacks considered in the existing machine-learning defensive deception approaches.

In Figs. 2.2 and 2.3, we summarized the number of game-theoretic or ML-based DD techniques that handled each type of attacks discussed in this paper. Note that the surveyed papers in this work were collected based on the following keywords: cyberdeception, DD, game-theoretic DD, and machine learning-based DD. The surveyed papers in this work were published from 1982 to 2020. As shown in Fig. 2.2, a majority of game-theoretic DD techniques considered node capture or compromise most and scanning (or reconnaissance) attack as the second most. As illustrated in Fig. 2.3, ML-based DD techniques considered spamming or malicious/fake profile attacks most as most ML-based approaches are mainly used to detect attacks in honeypots.

## 2.7 Application Domains for Defensive Deception

In this section, we provide how existing DD techniques have been studied in different network environments or platforms. To be specific, we described the characteristics of each network environment and corresponding DD techniques using game-theoretic or ML-based approaches, attacks considered, and metrics and experiment testbeds used.

## 2.7.1 Applications that are not Domain Specific

An enterprise network is a common system that is homogeneously configured to be operated in a static configuration [Kandula et al., 2009]. Hence, an attacker can easily plan and perform its attack successfully and accordingly penetrate the system without experiencing great difficulty but using dynamic strategies to defeat defense strategies.

#### DD Techniques

The common defensive deception (DD) techniques used in this network environment include various types of honey information, such as fake honey files [Whitham, 2016] or honeypots [Çeker et al., 2016; Pawlick and Zhu, 2015; Pawlick et al., 2018; Píbil et al., 2012; Rowe et al., 2006; Zhu et al., 2013].

#### Main Attacks

In enterprise networks, the following attacks have been countered by game-theoretic (GT) or ML-based DD, including insider threat and sophisticated attackers, such as APTs [Horák et al., 2017], zero-day attacks [Whitham, 2016, 2017], reverse engineering attacker using security patch Avery and Spafford [2017], or DoS attack [Çeker et al., 2016].

#### Key GT and ML Methods

For game-theoretic DD techniques deployed in this network environment, signaling games [Çeker et al., 2016; Pawlick and Zhu, 2015; Pawlick et al., 2018] or Stackelberg game [Zhu et al., 2013] have been commonly used. Some different game types, such as Metagames/Expected Utility [Píbil et al., 2012] or Bayesian Game using Bayesian equilibrium [Çeker et al., 2016] have been also considered to develop DD techniques in this network context. For ML-based DD techniques, Natural Language Processing (NLP) techniques [Whitham, 2016, 2017] have been used to develop honey files.

#### Pros and Cons

A majority of game-theoretic DD approaches have considered an enterprise network without providing the details of a network model with the aim of examining a proposed game-theoretic framework with solid theoretical analysis, such as identifying optimal strategies or solutions based on Nash Equilibria. However, highly theoretical game-theoretic DD approaches may not provide concrete details on how to design DD techniques considering the characteristics of each network environment in terms of resource availability, system security and performance/services requirements, or network and environmental dynamics. In addition, some general game-theoretic DD approaches do not specify a DD technique but can provide a general idea of modeling a DD technique. However, this lacks details, showing limited applicability in real systems. Further, enterprise networks are mostly highly complex systems that may need to deal with a wide range of attack types. Due to the abstract nature of game-theoretic approaches proposed for the enterprise networks, they do not provide how to simulate specific types of attacks.

## 2.7.2 General Cyber-Physical Systems (CPSs)

A CPS is a system that can provide communications between humans via cyber capabilities and physical infrastructure or platforms [Poovendran, 2010]. The CPS has been advanced with the cyber capabilities of communications, networking, sensing, and computing as well as physical capabilities with materials, sensors, actuators, and hardware. The CPS is uniquely distinguished from other platforms due to the presence of both cyber and physical aspects of systems and their coordination [Poovendran, 2010]. CPSs include wireless sensor networks, Internet of Things (IoT), software-defined networks (SDNs), and industrial control systems (ICSs) [Kathiravelu and Veiga, 2017; Serpanos, 2018]. Although we discuss each of them in separate sections, we include this section particularly to discuss DD techniques that are designed for "general CPSs" without specifying a particular platform.

#### DD Techniques

Although DD techniques have been deployed in CPS environments, their applicability seems not limited to only CPS environments; rather, they are applicable in any environment. We found bait information and honeypots are applied to the CPS environments for introducing confusion or uncertainty to attackers or luring them to honeypots for collecting attack intelligence [Hofer et al., 2019; Kiekintveld et al., 2015; Sayin and Başar, 2019].

#### Main Attacks

Game-theoretic or ML-based DD techniques in this domain mainly handled node compromise [Kiekintveld et al., 2015], scanning/reconnaissance [Sayin and Başar, 2019], or APT [Hofer et al., 2019] attacks.

#### Key GT and ML Methods

A signaling game theory was used for a defender to identify an optimal strategy using DD techniques [Sayin and Başar, 2019]. In addition, how to select honeypots to maximize confusion or uncertainty to attackers is also studied in [Kiekintveld et al., 2015]. To create decoy devices that look real, RNN (Recurrent Neural Networks) was also used based on observations of device behaviors for a year in a CPS [Hofer et al., 2019].

#### Pros and Cons

Since a CPS is a popular network environment and commonly used in real systems, developing DD techniques for the CPS can have high applicability in diverse CPS contexts. However, the DD techniques developed for the CPS did not consider much about its unique characteristics, such as the features of having both cyber and physical aspects and challenges of the environment itself. Hence, we do not observe many differences between the CPS's DD and the enterprise network's DD techniques. In addition, physical honeypots are sometimes recommended when dealing with highly intelligent attackers. However, such honeypot deployment and maintenance often come with higher deployment and management costs.

### 2.7.3 Cloud-based Web Environments

Cloud-based web applications become more common and popular than ever to deal with critical tasks that make security a prime concern. DD is one of the promising directions to defend against web attacks [Efendi et al., 2019]. DD techniques can be employed to defend against advanced web attacks that cannot be well handled by existing signature or anomaly-based intrusion detection or prevention mechanisms [Efendi et al., 2019].

#### DD Techniques

In this environment, honey-X [El-Kosairy and Azer, 2018] or fake signals [Pawlick et al., 2015] are used as DD techniques, such as honeywebs, honey files, honey tokens, or honeypots.

#### Main Attacks

Game-theoretic or ML-based DD techniques in cloud environments have mainly considered general web attacks [El-Kosairy and Azer, 2018], or multi-staged APT attacks [Pawlick et al., 2015].

#### Key GT and ML Methods

Signaling games with complete, imperfect information have been used to model attack-defense games [El-Kosairy and Azer, 2018; Pawlick et al., 2015].

#### Pros and Cons

While game-theoretic approaches proposed a generic game-theoretic DD framework [El-Kosairy and Azer, 2018; Pawlick et al., 2015], ML-based obfuscation technique is unique as it aims to defend against privacy attacks to deal with adversarial ML examples. However, we do not observe concrete design features of each DD technique only for cloud computing.

### 2.7.4 Internet-of-Things (IoT)

IoT network environments become highly popular and have been recognized as one of CPSs with the capability to provide effective services to users. IoT has been also specialized for particular domains such as Internet-of-Battle-Things (IoBT) [Kott et al., 2016], Industrial-Internet-of-Things (IIoT) [Pinto et al., 2017], or Internet-of-Health-Things (IoHT) [Rodrigues et al., 2018]. In addition, IoT embraces ideas of several CPS network environments, such as wireless sensor networks (WSNs), mobile ad hoc networks (MANETs), or smart city environments consisting of sensors and other intelligence machines [Kocakulak and Butun, 2017]. IoT has been popularly considered as a main platform for deploying cyberdeception technologies [Alshammari et al., 2020].

#### DD Techniques

As in other environments, honeypot-enabled IoT networks have been mainly considered by solving the optimal deployment of honeypots [Çeker et al., 2016; La et al., 2016].

#### Main Attacks

In IoT environments, game-theoretic or ML-based DD techniques aimed to defend against reconnaissance and probing attacks, DoS attacks [Çeker et al., 2016], packet dropping attacks [Çeker et al., 2016], or APT attacks [La et al., 2016; Xi and Kamhoua, 2020].

### Key GT and ML Methods

Bayesian games with incomplete information and meta games are considered for a player who is unsure of a type of attackers [La et al., 2016]. IoT in battlefields is referred to IoBT where deception games introduced in [Anwar et al., 2020; Anwar and Kamhoua, 2020; Anwar et al., 2019; Nan et al., 2019; Xi and Kamhoua, 2020]

### Pros and Cons

Since honeypots are fake nodes mimicking the behavior of a regular node, adding a honeypot does not change the hierarchy of the IoT network or the interface of IoT gateways. A game-theoretic honeypot technique is the only technique applied to this domain. However, the IoT naturally can generate a large amount of data for ML-based DD techniques which can create real-like decoys or enhance detecting inside and outside attackers.

Table 2.5: Application Domains of game-theoretic or Machine Learning-based Defensive Deception Techniques

| **Application domains** | **Defensive deception techniques** | **Main attacks** | **Key GT/ML techniques** | **Pros** | **Cons** |
|---|---|---|---|---|---|
| Cloud web-based environments: GT-Based [El-Kosairy and Azer, 2018; Pawlick et al., 2015] | Honeywebs (using honeytokens, honey files, decoy resources, honeypots), deceptive signals, and obfuscation | General web attacks, APT attacks, privacy attacks | Signaling game theory, a suite of ML classifiers | Multiple DD frameworks are provided with high applicability to deal with a wide range of web attacks | Some detailed designs should be considered to deal with unique challenges of cloud environments |
| Internet-of-Things: GT-Based [Anwar et al., 2020; Anwar and Kamhoua, 2020; Anwar et al., 2019; La et al., 2016; Nan et al., 2019; Xi and Kamhoua, 2020] | Honeypot | Reconnaissance and probing attacks, DoS attacks, packet dropping attacks, APTs | Bayesian games, meta games, signaling game with perfect Bayesian equilibrium | Since honeypots are fake nodes mimicking the behavior of a regular node, adding a honeypot does not change the hierarchy of the IoT network or the interface of IoT gateways. | A game-theoretic honeypot technique is the only technique applied to this domain. However, the IoT naturally can generate a large amount of data for ML-based DD techniques which can create decoys that look real or enhance detection of inside and outside attackers |

**Table 2.5 – (Continued)**

| **Application domains** | **Defensive deception techniques** | **Main attacks** | **Key GT/ML techniques** | **Pros** | **Cons** |
|---|---|---|---|---|---|
| Software-defined networks: GT-Based [Al Amin et al., 2019a, 2020; Chiang et al., 2018; Mao et al., 2019] | Honeypot | Reconnaissance attacks | Bayesian game | The key advantage of using SDN-based DD approaches is their easy deployability. For example, the existence of an SDN controller enables easily camouflaging a network topology or hiding vital nodes through flow traffic control. | Most existing approaches use a single SDN controller, which exposes a single point of failure |
| Wireless networks: GT-Based [Clark et al., 2012; Dimitriadis, 2007; Nan et al., 2020b] | Deceptive network flow, honeynet | Jamming attacks, 3G core network attacks | A noncooperative non-zero-sum static game | Specific design features to deal with key concerns of wireless networks are provided. | If a honeynet architecture is heavily dependent upon the accuracy of deployed honeypots, the formulated game may not work under different deployment settings |
| Online social networks: ML-Based [Badri Satya et al., 2016; Lee et al., 2010, 2011; Stringhini et al., 2010] | Social Honeypot, honey profiles | Fake liker (crowdturfing), social spammers, spamming | Supervised ML, ML toolkit, SVM | Learning the characteristics of attack behavior and attract spammers in OSN | Detectors are applied on static datasets to evaluate the detection of spammer and honey profiles are not generic and depend on the platform type |
| General Cyber-Physical Systems: GT-Based [Kiekintveld et al., 2015; Sayin and Başar, 2019]; ML-based [Hofer et al., 2019] | Crafted bait information, honeypot | Reconnaissance attacks, node compromise, APT | Signaling game theory, RNN | Due to a wide range of CPS applications, developing DD techniques will have high values and applicability in diverse CPS environments. | DD techniques for the CPS do not reflect the unique challenges of the CPS environments. Physical honeypot deployment and maintenance often come with higher deployment and management costs |

Table 2.5 – (Continued)

| Application domains | Defensive deception techniques | Main attacks | Key GT/ML techniques | Pros | Cons |
|---|---|---|---|---|---|
| Nondomain specific environment: GT-Based [Aggarwal et al., 2016, 2017; Al Amin et al., 2019b; Basak et al., 2019; Bilinski et al., 2019; Casey et al., 2016, 2015; Çeker et al., 2016; Cho et al., 2019; Durkota et al., 2015; Ferguson-Walter et al., 2019b; Garg and Grosu, 2007; Hespanha et al., 2000; Horák et al., 2017; Huang and Zhu, 2019; Mohammadi et al., 2016; Pawlick and Zhu, 2015; Pawlick et al., 2018; Píbil et al., 2012; Rahman et al., 2013; Shi et al., 2020; Thakoor et al., 2019; Wagener et al., 2009; Wang et al., 2020; Yin et al., 2013; Zhu et al., 2012, 2013]; ML-Based [Abay et al., 2019; Badri Satya et al., 2016; Ben Salem and Stolfo, 2012; El Idrissi Younes et al., 2016; Hofer et al., 2019; Krueger et al., 2012; Lee et al., 2010, 2011; Nanda et al., 2016; Stringhini et al., 2010; Whitham, 2016, 2017] | Honey files, honeypots, honeywebs, honeynets, honey patch, honey profiles, honey surface, honeybots, HMAC, obfuscation, deceptive signal, Fake Identities, deceptive network flow, social honeypots, bait-based deception, fake services | Zero-day attacks, APTs, masquerade attacks, reverse engineering for security patches, DoS attack, optimal inference attacks, reconnaissance attack, spam, social spam, malicious profiles | Signaling games, Stackelberg game, Metagames and Expected Utility, Bayesian Game using Bayesian equilibrium, subgame perfect Nash Equilibrium (SPNE), NLP techniques, hypergames, Stackelberg game, one-sided POSG, BayesNet, Decision Table, Naïve-Bayes, Reinforcement Learning, SVM | Most game-theoretic DD approaches propose a general deception game framework without specifying a certain domain environment, which can have high applicability regardless of domain environments. | Due to the nature of a general approach, there is high overhead to cover the general approach to a domain-specific approach |

### 2.7.5 Software-Defined Networks (SDNs)

The SDN paradigm separates data plane processing (e.g., packet forwarding) from control-plane processing (e.g., routing decisions) [MacFarland and Shue, 2015]. The OpenFlow protocol [Benton et al., 2013] acts as an API between network switches and a logically centralized decision maker, called the *OpenFlow controller*. In this protocol, the network switches cache data-plane flow rules. When a switch receives a packet and does not know how to forward it according to its cached rules, it sends an "elevation" request containing the original packet and a request for guidance to the controller. The controller examines the packet and sends a set of rules that the switch should add to the data plane cache for forwarding packets [Dixit et al., 2013]. deception techniques proposed in [Al Amin et al., 2019a, 2020] were designed to secure SDN.

#### DD Techniques

Honeypots are popularly used as a defense strategy in game-theoretic DD framework [Chiang et al., 2018; Mao et al., 2019] for taking dynamic, adaptive defense strategies.

#### Main Attacks

Common attack behaviors considered include scanning or reconnaissance attacks to obtain information and intelligence towards a target system by scanning network addresses (e.g., IP or port numbers) aiming to obtain defense information, network mapping, and inside information [Chiang et al., 2018]

#### Key GT and ML Methods

Bayesian game theory has been popularly adopted under various conditions [Mao et al., 2019], such as imperfect information, incomplete information, information sets, and perfect Bayesian equilibrium.

#### Pros and Cons

The key advantage of using SDN-based DD approaches is their easy deployability. For example, the existence of an SDN controller enables easily camouflaging a network topology or hiding vital nodes through flow traffic control. Most existing approaches use a single SDN controller, which exposes a single point of failure.

### 2.7.6 Wireless Networks

Wireless communications are everywhere these days and more common than wired communications due to their easy deployment and efficiency. However, when network resources are scarce, bandwidth constraints or unreliable wireless medium become main issues to be resolved. Game-theoretic or ML-based DD techniques are also proposed to defend against various types of attacks exploiting vulnerabilities of wireless network environments.

#### DD Techniques

A deceptive network flow is proposed to generate the flow of random dummy packets in multihop wireless networks [Clark et al., 2012]. A honeynet architecture addresses mobile network security (3G networks) [Dimitriadis, 2007] was developed to enhance the security of the core network of mobile telecommunication systems. In [Dimitriadis, 2007], a gateway was designed to control and capture network packets as well as investigate and protect other information systems from attacks launched from potentially compromised systems inside the honeynet. DD is used to transmit fake information over fake channels to mitigate jamming attacks in wireless networks [Nan et al., 2020b].

#### Main Attacks

Game-theoretic or ML-based DD techniques have defended against jamming attacks [Clark et al., 2012] in multihop wireless networks and 3G core network. The goal of attackers is to compromise servers or gateways that support nodes by providing general packet radio services [Dimitriadis, 2007].

#### Key GT and ML Methods

A noncooperative, non-zero-sum static game is used to model the interactions between an attacker and a defender [Dimitriadis, 2007]. Deceptive routing paths are also designed based on a two-stage game using Stackelberg game theory [Clark et al., 2012]. Deceptive power transmission allocation based on game theory is applied to defend wireless networks against jamming attacks [Nan et al., 2020b].

#### Pros and Cons

Specific design features to deal with key concerns of wireless networks, such as multihop communications or mobile wireless security, are helpful to implement real systems based on the given game-theoretic DD technologies. However, if a honeynet architecture heavily depends upon the accuracy of deployed honeypots, the formulated game framework may not guarantee the same level of effectiveness under different deployment settings as in Dimitriadis [2007].

### 2.7.7 Online Social Networks

Due to the large scales of OSNS and their significant influence in social, economic, and political aspects, AI and ML communities have recognized high challenges in developing DD techniques in the OSN platforms. Hence, there have been significant efforts made to secure OSNS from malicious users (e.g., fake users and spammers) by developing various social DD techniques [Badri Satya et al., 2016; Lee et al., 2010, 2011; Stringhini et al., 2010].

#### DD Techniques

The authors in [Badri Satya et al., 2016; Lee et al., 2010, 2011] specifically used social honeypots primarily to attract and identify attackers. Social honeypots can learn the behavioral models of malicious users and collect a spammer's preferences. Stringhini et al. [2010] analyzed 900 honey profiles for detecting spammers across MySpace, Facebook, and Twitter for classification and identification purposes.

#### Main Attacks

OSNS are mainly targeted by fake likers and spammers that would like to exploit the existing dynamics of OSNS to achieve a specific goal [Badri Satya et al., 2016; Lee et al., 2010, 2011; Stringhini et al., 2010].

#### Key GT and ML Methods

Traditional ML tool kits such as SVM have been used to analyze the data collected by social honeypots and honey profiles.

#### Pros and Cons

Social honeypots allow learning the characteristics of attack behavior and honey files can attract spammers to protect real files from them in OSN. However, detectors are applied to static datasets to evaluate the detection of spammers. In addition, honey profiles are not generic and should be customized to a specific platform type.

We summarized our discussions of this section in Table 2.5.

## 2.8 Evaluating Defensive Deception: Metrics and Testbeds

In this section, we survey what types of metrics are used to measure the effectiveness and efficiency of DD techniques. In addition, we address what types of testbeds are employed to evaluate the effectiveness and efficiency of DD techniques.

### 2.8.1 Metrics

In the literature, the following metrics are used to measure the **effectiveness** of existing DD techniques:

- *Detection accuracy* [Abay et al., 2019; Badri Satya et al., 2016; Basak et al., 2019; Ben Salem and Stolfo, 2012; Chiang et al., 2018; El Idrissi Younes et al., 2016; Lee et al., 2010, 2011; Pawlick et al., 2018, 2019; Xu et al., 2012; Yang et al., 2014]: This is often measured by the AUC (Area Under the Curve) of ROC (Receiver Operating Characteristic). AUC is used to measure the accuracy of detection by showing TPR (True Positive Rate) as FPR (False Positive Rate) increases. Abay et al. [2019] used classifier accuracy as a metric to evaluate the effectiveness of the developed Honey data to deceive the attacker. Badri Satya et al. [2016] and Chiang et al. [2018] used an algorithm to discover fake Liker in social networks. The authors in [Ben Salem and Stolfo, 2012; Xu et al., 2012] evaluated a masquerade attack detector based on AUC. ROC is also used to measure detection accuracy of social honeypot-based approach against social spammers [Lee et al., 2010, 2011] and malicious account [El Idrissi Younes et al., 2016; Yang et al., 2014] in online social networks. Pawlick et al. [2018, 2019] evaluated their deception mechanism based on different metrics, including the detector accuracy developed by the attacker to discover deception.

- *Mean time to detect attacks* [Basak et al., 2019]: The effectiveness of a detection mechanism is also measured by how early their deception technique can capture the attacker.

- *Utility* [Anwar et al., 2020; Anwar and Kamhoua, 2020; Anwar et al., 2019; Casey et al., 2016; Clark et al., 2012; Garg and Grosu, 2007; Horák et al., 2017; Huang and Zhu, 2019; Kiekintveld et al., 2015; Milani et al., 2020; Mohammadi et al., 2016; Nan et al., 2019, 2020a,b; Pawlick et al., 2018, 2019; Píbil et al., 2012; Thakoor et al., 2019; Tsemogne et al., 2020; Wang et al., 2020; Xi and Kamhoua, 2020; Zhang et al., 2019; Zhu et al., 2012]: In game-theoretic DD techniques, a player (either an attacker or a defender)'s utility (or payoff) is considered as one of key metrics to evaluate a deception game between the attacker and defender. Commonly, the utility of taking a DD strategy is formulated based on the deployment cost and the confusion increased to an attacker. Some signaling games model the interactions between an attacker, a benign client, and a defender. As a result, some of these studies considered the impact to normal as another cost of deception technology [Rahman et al., 2013; Wang et al., 2020]. A defender's expected utility is a common metric to be maximized as the system's objective, as shown in game-theoretic or ML-based DD techniques in [Al Amin et al., 2019a, 2020; Garg and Grosu, 2007; Horák et al., 2017; Huang and Zhu, 2019; Kiekintveld et al., 2015; Milani et al., 2020; Nan et al., 2020a; Pawlick et al., 2018, 2019; Píbil et al., 2012; Thakoor et al., 2019; Tsemogne et al., 2020; Xi and Kamhoua, 2020; Zhang et al., 2019; Zhu et al., 2012].

- *Probabilities of an attacker taking certain actions* [Bilinski et al., 2019; Cranford et al., 2020, 2019; Nan et al., 2019; Rahman et al., 2013; Wagener et al., 2009]: An attacker's actions in proceeding attacks are also considered as a metric to measure the effectiveness of DD (e.g., honeypots). For example, in [Wagener et al., 2009], the following probabilities of an attacker's action are considered: a probability of an attacker retrying a failure command, a probability of an attacker choosing an alternative strategy, and a probability of an attacker leaving a game. The probability of an attacker to successfully control the network and overcome the deception is used to evaluate the effectiveness of cyberdeception in [Bilinski et al., 2019; Rahman et al., 2013] and the probability of launching an attack as in [Cranford et al., 2020, 2019].

In addition, we also found the following metrics to capture the **efficiency** of the surveyed DD techniques:

- *Round trip-time* [Aggarwal et al., 2019; Araujo et al., 2014; Basak et al., 2019; Borello and Mé, 2008; Cai et al., 2009; Chan and Yang, 2004]: The key role of DD is to delay the attacker processes [Araujo et al., 2014]. Increasing the time required to crack an obfuscated code is the evaluation metric adopted in [Cai et al., 2009; Chan and Yang, 2004], as well as the complexity of the deobfuscation process [Borello and Mé, 2008]. The HackIt deception tool in [Aggarwal et al., 2019] successfully increased the time taken by hackers to exploit a system. An attacker's deception detection time is of great significance in evaluating the effectiveness of cyberdeception [Basak et al., 2019].
- *Runtime* [Avery and Spafford, 2017; Avery and Wallrabenstein, 2018; Shi et al., 2020]: This evaluates the cost of automated deception and obfuscation techniques. The performance of the deception technique based on fake patches [Avery and Spafford, 2017; Avery and Wallrabenstein, 2018] was evaluated using this metric. Similarly, the learning time also evaluates the learning the attacker's behavior model from attack data introduced in [Shi et al., 2020].

In Figs. 2.4 and 2.5, we summarized the types and frequency of metrics measuring performance and security of the surveyed DD techniques. As shown in Fig. 2.4, a player's utility is the most common metric among all metrics used for the surveyed game-theoretic DD techniques. On the other hand, in Fig. 2.5, ML-based DD approaches heavily relied on detection accuracy using such metrics as AUC, TPR, and FPR metrics.

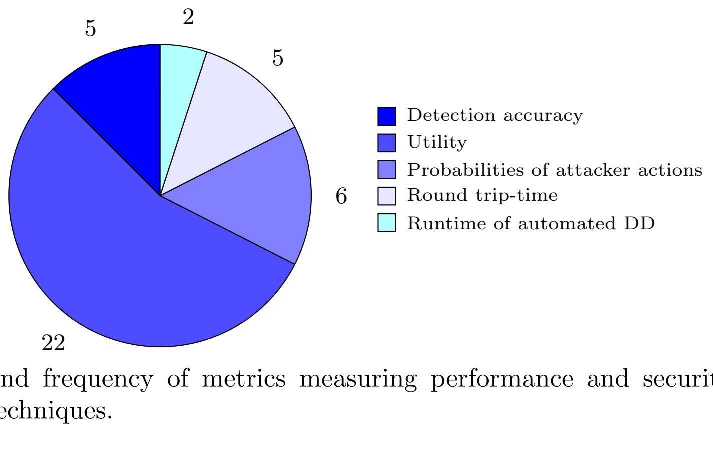


Figure 2.4: Types and frequency of metrics measuring performance and security of game-theoretic defensive deception techniques.

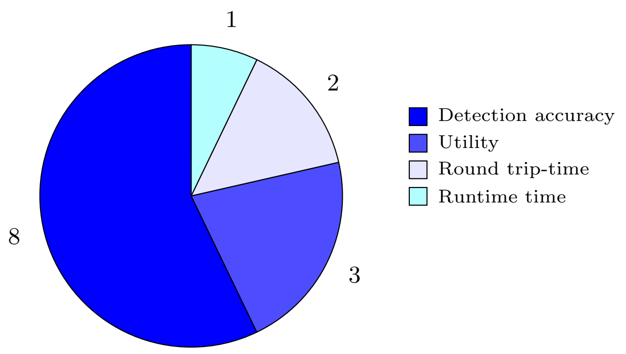


Figure 2.5: Types and frequency of metrics measuring performance and security of ML-based defensive deception techniques.

### 2.8.2 Evaluation Testbeds

In this section, we classify evaluation testbeds of the existing game-theoretic or ML-based DD techniques surveyed in this work in terms of the four classes: those based on probability, simulation, emulation, and real testbed. We discuss each of the evaluation testbeds and discuss the general trends observed from the survey.

#### Probability Model-Based Evaluation

Stochastic Petri Nets (SPN) probability models was used to evaluate the performance of an integrated defense system that combines multiple deceptive defense techniques [Cho and Ben-Asher, 2018]. In addition, the SPN is used to measure the performance of the deceptive defender and the attacker in a hypergame model [Cho et al., 2019].

#### Simulation Model-Based Evaluation

To simulate benign network traffic, Rahman et al. [2013] generated network flow by the Internet Traffic Archive [Danzig et al., 2008], using *Nmap* to test their deception technology. Chiang et al. [2016, 2018] simulate their deception system and combine it with their proposed work. Thakoor et al. [2019] used the CyberVan [Chadha et al., 2016; Pham et al., 2020] as a network simulation testbed to simulate their game-theoretic cyberdeception techniques. Al Amin et al. [2019b] and Bilinski et al. [2019] proposed a deceptive system which was simulated using a Markov decision processes. Several studies also simulated ML-based deceptive algorithms or classifiers to examine its accuracy [Abay et al., 2019; Al Amin et al., 2019a,b, 2020; Al-Shaer et al., 2019; Badri Satya et al., 2016; Ben Salem and Stolfo, 2012; Shi et al., 2020]. A masquerade attack is simulated to evaluate the developed detection techniques based on deception [Ben Salem and Stolfo, 2012; Xu et al., 2012]. Various types of game-theoretic based deception framework have been evaluated based on simulation models [Anwar et al., 2020; Anwar and Kamhoua, 2020; Anwar et al., 2019; Avery and Spafford, 2017; Basak et al., 2019; Bilinski et al., 2019; Cadar et al., 2008; Chiang et al., 2018; Clark et al., 2012; Garg and Grosu, 2007; Horák et al., 2017; House and Cybenko, 2010; Huang and Zhu, 2019; Kiekintveld et al., 2015; Kulkarni et al., 2020; La et al., 2016; Milani et al., 2020; Mohammadi et al., 2016; Nan et al., 2019, 2020a; Pawlick et al., 2018; Píbil et al., 2012; Rahman et al., 2013; Thakoor et al., 2019; Tsemogne et al., 2020; Xi and Kamhoua, 2020; Yin et al., 2013; Zhang et al.,

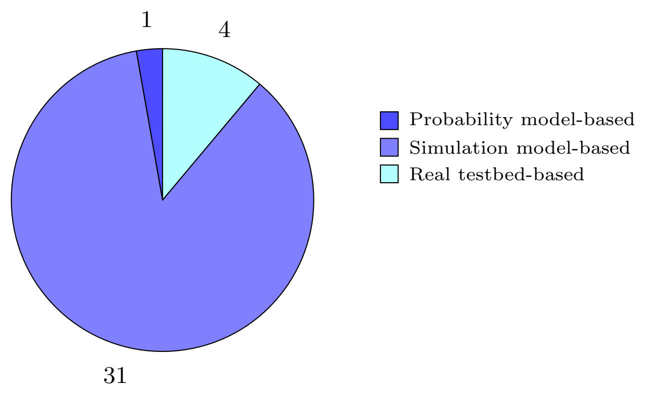


Figure 2.6: Evaluation testbeds for game-theoretic defensive deception techniques.

2019]. Simulation-based models showing a proof of concept have been developed to demonstrate the runtime of the proposed automated deception algorithm [Avery and Spafford, 2017; Cadar et al., 2008].

### Emulation Testbed-Based Evaluation

Emulation network environments were proposed to dynamically place deception resources [Wang et al., 2020], deploy code obfuscation techniques on real-like decompilers [Chan and Yang, 2004], and deploy DD techniques in emulated SDN-based data center network testbed.

### Real Testbed-Based Evaluation

An application for data collection has been developed [Aggarwal et al., 2016, 2017, 2019]. Experiments with human-in-the-loop with human participants (e.g., an attacker or a defender) have been conducted [Aggarwal et al., 2016, 2017, 2019], such as HackIt to evaluate the effectiveness of deception on participating hackers under different deception scenarios. Socialbots on Twitter real networks are significantly leveraged to detect and study content polluters and understand how spammers choose their spamming targets to ultimately create a malicious profile classifier [El Idrissi Younes et al., 2016; Lee et al., 2010, 2011; Yang et al., 2014]. In addition, various DD techniques are validated under real settings, such as honey-patches-based deception system [Araujo et al., 2014], location obfuscation for preserving users' privacy [Ardagna et al., 2007], and PRISMA (PRotocol Inspection and State Machine Analysis) tool capable of learning and generating communication sessions for deception [Krueger et al., 2012].

In Figs. 2.6 and 2.7, we summarized the frequency of particular testbeds used in developing game-theoretic and ML-based DD techniques surveyed in this chapter, respectively. The overall trends observed were that simulation-based evaluation was dominant in both GT and ML-based DD approaches. Although ML-based DD techniques are less explored than GT-based DD techniques, it is noticeable that more than 40 percent of the ML-based DD techniques surveyed here used evaluation based on a real testbed. This is natural that ML-based approaches basically need actual datasets to analyze. Another noticeable finding is that there has been a lack of probability model-based approaches. This is because it is highly challenging to model a complex system environment characterized by many design parameters based on mathematical models.

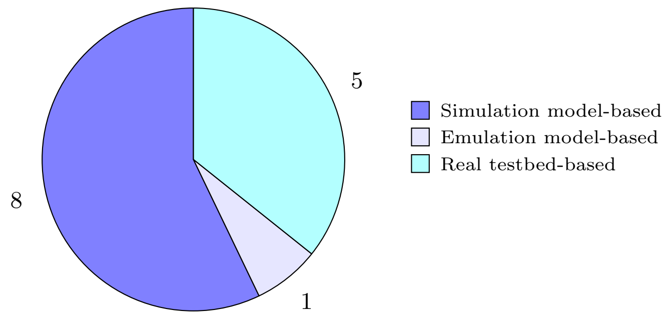


Figure 2.7: Evaluation testbeds for ML-based defensive deception techniques.

## 2.9 Conclusion

In this section, we discussed insights and lessons learned and limitations from the extensive survey conducted in this work. In addition, based on the conducted survey, we provided a set of promising future research directions.

### 2.9.1 Insights and Lessons Learned

We now revisit and answer the research questions raised in Section 2.1.4.

**RQ Characteristics:** *What key characteristics of DD distinguish it from other defensive techniques?*

**Answer**: Unlike traditional defense mechanisms, deception requires a certain level of risk as it requires some interactions with attackers with the aim of confusing or misleading them. Particularly, if the goal of defense requires long term deception, it is inevitable to face risk. In this case, DD can be used with other legacy defense mechanisms such as intrusion prevention or detection mechanisms to avoid too high risk. Moving target defense (MTD) or obfuscation techniques share a similar defense goal with DD, such as increasing confusion or uncertainty for attackers. However, unlike MTD or obfuscation which changes system configuration or information based on the existing resources of a system, DD can create false objects or information with the aim of misleading an attacker's cognitive perception or forming a misbelief for the attacker to choose a sub-optimal or poor attack strategy.

**RQ Metrics:** *What metrics are used to measure the effectiveness and efficiency of the existing game-theoretic or ML-based DD techniques?*

**Answer**: As surveyed in Section 2.8 along with Figs. 2.4 and 2.5, the effectiveness of GT-based deception is mainly measured based on utility. On the other hand, ML-based deception is mainly evaluated according to the accuracy of classifiers or intrusion detectors. The existing metrics observed in the surveyed literature for game-theoretic or ML-based DD approaches do not capture the direct impact of DD. That is, they do not quantify the uncertainty or confusion induced by the different deceptive techniques. Moreover, new metrics should be developed to measure the effectiveness of deception in misleading the attackers, such as lightweight deception with minimal or low defense cost in deploying or maintaining a specific deception technique.

According to Section 2.8 along with Figs. 2.4 and 2.5, GT-based deception techniques have been mainly evaluated in terms of the defender's utility. However, ML-based deception techniques focused on improving the accuracy of classifiers or intrusion detectors. However, the existing evaluation of GT-based DD techniques rarely consider utility analyses based on the losses and gains in terms of the timing of using different types of deceptions. In addition, although it is important to use metrics capturing deception cost introduced by its deployment and maintenance and performance degradation caused by running deception techniques, there has been much less effort to use those metrics. Furthermore, measuring the extent of attackers deviating from the optimal course of actions can be another meaningful metric to evaluate the effectiveness of games with DD.

**RQ Principles:** *What key design principles help maximize the effectiveness and efficiency of DD techniques?*

**Answer**: It is critical to make three key design decisions, which are what-attacker-to-deceive, when-to-deceive, and how-to-deceive. These three design decisions should be determined based on critical tradeoffs between effectiveness and efficiency of a developed DD technique. The what-attacker-to-deceive question should be answered based on what attacks are targeted by a defender and what attacks should be commonly handled in each application domain because each application domain generates distinctive challenges. When-to-deceive is related to the fact that depending on when a DD technique is used to deal with attackers in a certain stage of the cyber kill chain, the extent of deployability, effectiveness, efficiency, and cost of using a DD technique can be affected. For example, when-to-deceive can be decided according to how complex the deployment of such DD technique or how effective the DD technique is in terms of the timing. In addition, it can consider how costly deception is in terms of the availability of the resources. In some settings, deception is needed only temporarily to delay an attack until a defense system needs to take enough time for its reconfiguration or deploying/performing a certain defense operation (e.g., deceive an attacker until the MTD is executed to fully reconfigure an OS). Each type of a DD technique has its own unique characteristics and corresponding expected outcomes. Given what attack to deal with by determining what-to-deceive and when to deceive the attacker (i.e., in the attack stage), the how-to-deceive question is related to answering (1) what DD techniques are better than other DD techniques under the available resources and (2) what quality of deception is most appropriate to deal with specific attacks? For example, if a defense system can afford to deploy high-quality honeypots to deal with highly intelligent attackers, high-interaction honeypots would be a good choice. However, if the defense system has a limited budget or resources available and wants to deploy a lightweight DD technique temporarily (e.g., before a new setting of MTD is fully reconfigured), a honey file or honey token can be used with low cost but they can be detected by intelligent attackers soon. Therefore, depending on the defense cost budget, the system's resource availability, or the expected outcome for a DD technique to be used, a different type of DD techniques with a different level of deception quality can be adopted to meet the expected effectiveness and efficiency.

**RQ GT:** *What are the key design features when a DD technique is devised using game theory?*

**Answer**: Game-theoretic DD techniques mainly adopt an attack-defense framework where the attacker and defender interact with a conflict of interest scenario, which often use a form of zero-sum games. However, as multiple players in each party (i.e., multiple attackers for collusive attacks or multiple defenders for cooperative defense) can be involve in a attack-defense game, general-sum games are often considered as well. To be more specific, we also summarize a game type used in each paper that proposes game-theoretic DD in Table 2.3. Game-theoretic DD focuses on identifying optimal defense strategies to confuse or mislead an attacker by increasing uncertainty. Three design features are: (i) each player needs to have a clear goal and a corresponding clear utility function that reflects the player's intent, tactics, and resources; (ii) modeling and simulating how to increase confusion or uncertainty via different types of DD techniques should be elaborated; and (iii) metrics to measure effectiveness and efficiency should be articulated.

**RQ ML:** *What are the key design features when a DD technique is developed using ML?*

**Answer**: ML-based DD techniques learn and detect attack behaviors. However, ML can improve DD by considering the following: (i) We need to consider what types of datasets use to develop DD techniques. To develop believable fake objects presupposes good datasets for mimicking real objects and evaluating fake objects. Along these lines, targeted attack datasets should be used to model attackers and develop honey resources; (ii) ML-based DD approaches should adopt relevant metrics to capture their effectiveness and efficiency. So far, classification accuracy is the only metric to capture ML-based honeypots. Additional metrics should be developed to capture the quality of ML-based DD techniques; and (iii) A fully automated deception is highly desirable for ML-based DD techniques, such as generating deceptive traffics and network topologies.

**RQ Applications:** *How should different DD techniques be applied in different application domains?*

As the enterprise network is a complex domain, multiple honey-X techniques, possibly in combination, are applicable. For simpler domains, an easy-to-deploy honeypot is effective and widely used. However, developing high quality DD techniques often incurs more cost or higher complexity. For example, deploying more honeypots may increase the chance to mislead an attacker. However, too many honeypots also increase unnecessary costs and confuse the administrator or legitimate users. Depending on applications, DD techniques should be applied in an automated manner. For example, online social networks require deception for detecting attacks or collecting attack intelligence since social spammers do not aim to gain control but to influence other users in the network.

As you can observe in our extensive survey, many more game-theoretic DD studies have been conducted while ML-based DD studies are limited to developing honey-X-based approaches aiming to detect more attacks. Via our extensive survey, we could clearly observe the main benefit of using game-theoretic DD techniques is their capability to allow players to make strategic decisions and take various strategies upon observed dynamics of a game over time. In addition, since game theory provides a mathematical framework to model the interactions between attackers and defenders, it can be more extensively applied in developing various types of DD techniques as a general framework. However, as we also observed in existing work, although game-theoretic DD has powerful capability with solid mathematical proofs in developing DD techniques, their validation in security and performance has been heavily limited to the theoretical or simulation-based verification. However, ML-based DD approaches have been applied to mainly honey-X applications, mostly developing honeypots. Unlike other DD techniques, the main goals of honeypots are two-fold: protecting system assets by creating fake objects, which are honeypots and detecting attackers by luring them to the honeypots. Although the original purpose of DD more focuses on protecting the system assets, which is more like passive defense, the effectiveness of honeypots has been mainly measured its role in detecting attacks in terms of the detection accuracy. That is, ML-based honeypots have been developed by creating a fake object which looks like real with the aim of attracting more attackers. Compared to GT-based DD approaches, ML-based DD approaches are less dynamic because ML techniques are mainly used to improve the quality of deception in its development time and as a classifier to detect attacks. Although reinforcement learning as one of promising ML techniques as been used to develop a DD technique along with game theory [Al Amin et al., 2019b; Wang et al., 2020], hybrid approaches incorporating ML into game theory to develop DD techniques are rarely taken. We discuss examples of how these can be incorporated into a hybrid DD technique in the future research directions discussed in Section 6.

### 2.9.2 Limitations of Current Defensive Deception

From our extensive survey on the state-of-the-art defensive deception (DD) techniques using game theory and ML, we found the following limitations of the existing approaches:

- Although many DD techniques have used game-theoretic approaches, they have mainly focused on theoretical validation based on Nash equilibrium and the identification of optimal solutions, in which attacks are not often detailed enough but simply considered attacks compromising other nodes. Often, attacks are complicated and performed with more than one episode or multistage over time, such as APT attacks. Therefore, although we can benefit from modeling simple attack processes, such as active reconnaissance or insider attack, it is still challenging to deploy the game-theoretic DD in real systems and model attacks based on a complex cyber kill chain.
- We observed that an enterprise networking is the only domain that is well developed. Research effort in other domains has been limited and mainly focused on reconnaissance attacks. IoT and SDN environments can generate large amounts of traffic flow data, which can be used to train ML models to identify attackers. However, current DD approaches for those domains lack using ML.
- Due to high complexity of action spaces, most game-theoretic DD has dealt with a limited set of actions for an attacker and defender, which cannot accurately model an attack or defensive process when the attacker can perform multiple attacks following the multiple stages of attack processes, such as cyber kill chain in APT attacks.
- When multiple attacks arrive in a system, considering the interactions only between one attacker and one defender is not sufficient to capture the practical challenge. How to interpret a response towards a certain action (e.g., who is responding to what action when there are multiple attackers) is not clear although it is critical to model the interactions between the attacker and defender.
- A majority of game-theoretic DD approaches are studied at an abstract level by using game-theoretical analysis without explicitly addressing design challenges derived from the network environment or platform the DD technique is deployed. When some ideas are adopted and applied in a certain network environment, it is quite challenging to deploy a game-theoretic work onto a specific platform due to a lack of detail regarding the deployment process.
- The accuracy of ML-based deception techniques depends on data availability regarding the attacker identity, techniques, and targets. Practically, such data are not available for the defender, which significantly limits training ML-based classifiers or detectors. Moreover, such models tend to assume that the attacker is acting normally toward its target. However, if an attacker decided to deceive a defender to remain stealthy, the effectiveness of DD techniques may not be clearly measured.

- Compared to game-theoretic DD techniques, ML-based DD has been much less studied. The majority of ML techniques in DD are used to detect attacks in honeypots. Only a few works have used ML to mimic real objects or information for developing honey objects or information.

- Although game-theoretic or ML-based DD has been studied, there have been a few studies that combine both to develop hybrid DD techniques, particularly incorporating reinforcement learning (RL) into players' utility functions and using RL to identify the players' optimal strategies.

- Evaluation metrics of DD are limited. Most game-theoretic DD approaches mainly used metrics in game theory, such as utility or probability of taking attack strategies. Most ML-based DD approaches were mainly studied to achieve attack detection, which leads to using detection accuracy as a major metric. These metrics, utility and detection accuracy, do not fully capture the effectiveness and efficiency of DD approaches.

- Evaluation testbeds are mostly based on simulation models. Although some ML-based DD approaches used real testbeds, they are mostly social honeypots deployed in social media platforms. In particular, game-theoretic DD approaches remain highly theoretical analysis.

CHAPTER

3

# GAME THEORY AND MACHINE LEARNING-BASED ADAPTIVE HONEYFILE SYSTEM FOR INSIDE ATTACKER DETECTION

Insider attacks pose a major threat to computer security and current cybersecurity approaches are inadequate in dealing with them. Advanced Persistent Threats (APTs) involve an attacker masquerading as a legitimate user and carrying out malicious activities such as data exfiltration. Traditional cybersecurity techniques, such as firewalls, cryptography, and intrusion detection systems, are ineffective against such attacks. Defensive deception is a new way to detect and mitigate insider attacks. Honeyfiles are a popular defensive deception approach that involves placing files on devices to confuse and detect insider attackers. However, honeyfiles can generate too many false-positive alarms, which can confuse legitimate users.

In this chapter, we have developed a novel honeyfile approach called *Mee*, which adjusts the number and placement of honeyfiles according to the risk profiles of the devices. We have modeled our approach as a honeyfile game to help the defender search for an optimal strategy that determines the number of honeyfiles in each connected device, considering the probability distribution of the attacker's targets. Our experimental results show that Mee reduces false positive alarms from legitimate users and increases attacker costs.

Through the use of Bayesian games, the defender can find the optimal solution. However, Mee only considers a relatively simple scenario with only two players at a time, and many details about users and devices are ignored for simplicity. To improve Mee, we propose a honeyfile system called HoneyMee that applies Deep Reinforcement Learning (DRL) to automatically detect and respond to malicious activities. HoneyMee considers all connected devices and records any alarms triggered when someone interacts with a honeyfile. It then evaluates the alarms while examining the typical behavior of legitimate users and the security status of the environment. We simulate the interactions between legitimate users, insider attackers, and defenders. Our findings demonstrate that HoneyMee significantly improves the accuracy of insider threat detection and the payoff for the defender.

## 3.1 Introduction

Recently, much attention has been paid to the issue of insider threats. Traditional defensive strategies are inadequate to detect and respond to insider attacks. According to Salem et al. [2008], insider threats can be divided into two categories: *traitors*, who exploit their valid credentials, and *masqueraders*, who pretend to be a legitimate user (and usually have less knowledge than a traitor about where the victim's valuable data is located).

An advanced persistent threat (APT) is a way for an attacker to disguise themselves. Chen et al. [2014] have identified six stages of an APT attack: (1) reconnaissance; (2) delivery; (3) initial intrusion; (4) command and control (C2); (5) lateral movement; (6) data exfiltration. During the first three stages, the attacker collects user data, such as accounts and passwords, through phishing and creates backdoors on a compromised device. This enables them to remotely control and access the victim's device without being detected by traditional cybersecurity technologies, such as IDSs and firewalls. Recent research has focused on blocking APTs [Marchetti et al., 2016; Shan-Shan and Ya-Bin, 2018; Tankard, 2011]. In contrast, our research focuses on the detection of masquerader-type insider attacks.

Defensive deception offers a fresh approach to identifying and disrupting attacks that are not effectively countered by traditional cybersecurity methods such as intrusion detection systems and firewalls. Rather than solely focusing on the actions of attackers, defensive deception involves actively participating and predicting their potential actions [Almeshekah and Spafford, 2014]. In their study, Zhu et al. [2021b] investigate the advancements in defensive deception technologies through the examination of game theory and machine learning-based research. Defensive deception [Lu et al., 2020] is a viable strategy for mitigating and detecting various types of attacks, including reconnaissance [Achleitner et al., 2016; Anjum et al., 2021] and insider attacks [Ben Salem and Stolfo, 2012; Bowen et al., 2009]. [Almeshekah and Spafford, 2014]. The goals of deception are twofold: to confuse attackers by wasting their efforts and concealing sensitive information, and to detect malicious actions.

Honeyfiles, or decoy documents, are a lightweight defensive deception technology [Bowen et al., 2009; Voris et al., 2015; Yuill et al., 2004] comprising two main tasks: 1) generate content in a honeyfile [Abay et al., 2019; Whitham, 2016, 2017], and 2) decide the number and placement of honeyfiles [Bowen et al., 2009; Gómez-Hernández et al., 2018; Voris et al., 2015]. Bowen et al. [2009] propose the *Decoy Document Distributor ($D^3$) system*, which generates and places decoy documents in a file system. Salem and Stolfo [2011] analyze legitimate users and attackers that are affected by the number and location of honeyfiles. Increasing the quantity of honeyfiles on a device enhances the likelihood of confusing and detecting the attacker. However, this also results in higher resource consumption costs and a higher false-positive rate in detecting attackers, as legitimate users may be confused.

In this chapter, we introduce Mee, a specialized honeyfile approach designed for large-scale enterprise networks. Mee demonstrates a decentralized deployment strategy, enabling any user to implement it. Additionally, it demonstrates a centralized control mechanism, where the defender evaluates suspicious activities throughout the network to determine the number and placement of honeyfiles for each device. This approach assists the defender in achieving the following objectives: detecting ongoing attacks and identifying compromised devices. Since hackers infiltrating a device's file system will need to search it for valuable information, they are more likely to interact with a honeyfile compared to legitimate users who are familiar with their file systems.

Our contributions include the following:

- We introduce Mee, a fresh honeyfile system designed to identify insider attackers within enterprise networks. Mee has the capability to adaptively modify the quantity of honeyfiles deployed on each device.
- We present a Bayesian game model that Mee can use to examine the most advantageous strategies for both the attacker and the defender.
- To enhance the Mee research, we propose the introduction of HoneyMee, which is a honeyfile system based on deep reinforcement learning (DRL).
- A DRL model is developed to aid the defender in analyzing the environment, comprising attackers and users, in order to determine the most effective strategy.
- We conduct simulations and compare Mee and HoneyMee with the conventional honeyfile approach. Our results demonstrate that Mee and HoneyMee outperform the honeyfile approach in detecting insider attackers and also have a lesser impact on legitimate users.

### Structure of This Chapter

The rest of this chapter is organized as follows. Section 3.2 compiles previous research on honeyfile as well as studies on defensive deception utilizing game theory or machine learning approaches. Section 3.3 describes our problem in terms of details of the insider attackers and the threat model. Section 3.4 presents the concept of Mee, a Bayesian game-based honeyfile system. This section covers the design of Mee, the scenario and models used in the research, as well as the implementation and simulation of Mee. Section 3.5 expands upon the preceding section by incorporating Deep Reinforcement Learning (DRL) to address a more complex environment and provide additional insights into the characteristics of attackers, users, and the defender. Section 3.6 describes the conclusions and our future directions.

## 3.2 Related Work

As suggested by Yuill [2007], defensive deception involves the defender creating tactics to deceive the attacker into taking or avoiding certain steps to improve the system's security. Almeshekah and Spafford [2016] further developed the concept of cyberdeception by introducing the concept of *confusion* and redefining it as "planned actions taken to mislead or confuse attackers and thus cause them to take (or not take) specific actions that aid computer security defenses." Zhu et al. [2021b] survey current deception technologies. They explore game theory and machine learning-based deception studies and assess their advantages and disadvantages.

The honeyfile system is an intrusion detection approach based on deception. Yuill et al. [2004] proposed a file server with a honeyfile system to confuse and detect threats. Gómez-Hernández et al. [2018] developed a honeyfile-based security system, called R-lock, to protect against ransomware. Whitham [2017] designed a high-interaction system that can generate a honeyfile to mimic a selected document. Salem and Stolfo [2011] conducted an empirical study on decoy documents, asking students to download and install them. Through monitoring the students' behaviors, they determined the most effective placement and naming of the decoy documents to reduce false positives.

Game theory applies well in defensive deception [Pawlick et al., 2019; Zhu et al., 2021a]. Pawlick et al. [2018] use signaling games to model interactions between the defender and APT attackers. La et al.

[2016] propose a two-player game to model attacker and defender in an IoT environment with honeypots used for defensive deception. Cho et al. [2019] use a hypergame model to show how differences in perceptions of players affect decision-making.

Deep Reinforcement Learning (DRL) has been successful in teaching the agent to look for tactics that will maximize the agent's reward. [Gu et al., 2016; Hester et al., 2018; Mnih et al., 2015]. An agent interacts with its environment, attempting to learn the best possible policy to make sequential decisions through trials. The utilization of DRL has revolutionized the field of artificial intelligence by allowing for direct learning from high-dimensional raw input without the requirement of manually crafted features. DRL is widely used in cybersecurity [Nguyen and Reddi, 2023] and defensive deception [Li et al., 2022; Olowononi et al., 2022]. Utilizing DRL in the training process, the defenders can search for the most effective policy that will make them more accurate and save effort when defending against attackers.

## 3.3 Problem Statement

Our focus is on masquerader-type insider attackers operating within an enterprise network environment. These attackers have the ability to obtain authorization from device owners, allowing them to search for valuable files on target devices without triggering traditional defensive technologies like Intrusion Detection Systems (IDS).

In the context of deployed honeyfiles, the attacker may encounter one of three outcomes after successfully infiltrating a device.

**Success:** Viewing or transferring valuable files from a device.

**Invalid:** Not finding a valuable file, that is, suffering wasted effort but no additional loss.

**Defeat:** Triggering alarms by touching honeyfiles. The defender would update its belief about the network's security level while cleaning or replacing the compromised device, as a result of which the attacker would lose a compromised device and have wasted its effort.

Although an attacker is capable of gaining unauthorized access to devices, it lacks knowledge about the location of valuable files on these devices. An attacker performs reconnaissance before attacking to investigate whether a device contains valuable files to save energy. Thus, an insider attacker has a clear intent regarding what information is valuable and targets devices accordingly. For example, an attacker who sought a final exam in a university would target several specific professors in a particular department, not a random device. Therefore, we name the device owner's identity, such as professor, student, and CEO, as *organizational role*. An insider attacker may pay more attention to several specific organizational roles. An insider attacker may focus on certain organizational roles, increasing the probability that honeyfiles on the associated devices will be activated. Additionally, insider attackers are less familiar with file systems than legitimate users which means that they must put in more effort to explore the devices and have a higher chance of triggering honeyfiles.

We assume that the attacker is aware of the presence of honeyfiles, but cannot differentiate them from regular files. For the defender, honeyfile alarms are the only way to identify a potential insider attacker. Thus, the defender must determine whether the alarms are from users or insider attackers. We assume that legitimate users have certain patterns of behavior. For instance, a user may be more likely to use his device during working hours than at night. Furthermore, being familiar with their devices, legitimate users are more likely to avoid honeyfiles, but there is still a chance that they may accidentally interact with them.

## 3.4 Mee

The use of honeyfiles has two purposes: to confuse the attacker with false information and detect any unauthorized access to connected devices. The number of honeyfiles in one device can significantly change the effectiveness of a honeyfile system. However, current honeyfile research mainly focuses on the honeyfile system in a single device, which cannot provide adequate observation to assist the defender in estimating the security situation and changing the number of honeyfiles. In contrast, we consider all devices in an entire enterprise network. Specifically, we have a central controller, named *Mee controller*, that receives information (e.g., about accesses to honeyfiles) from and instructs clients, named *Mee client*, which resides on individual devices. Following instructions from the Mee Controller, Mee client increases or decreases the number of honeyfiles.

### Mee Client

The Mee client is a software component that is installed on each device. It is responsible for generating and deleting honeyfiles, as well as detecting suspicious activities related to them, such as opening, modifying, deleting, and transferring. Whenever someone interacts with a honeyfile, the Mee client sends an alert to the Mee controller and awaits instructions. To ensure that honeyfiles are not mistaken for legitimate files and to draw the attention of the attacker, they must be placed and named appropriately.

Salem and Stolfo [2011] conducted an empirical study on decoy documents, in which they asked 52 students to download and install decoy documents and monitored their behavior to determine the best placement and names for decoy documents that would minimize false positives. Although our focus is mainly on the number of honeyfiles on each device, we also take into account existing research on honeyfile names that are more likely to be of interest to the attacker. Therefore, we assign a *sensitivity* value, $H_s \in [1, 3]$, to each honeyfile, with higher values indicating greater attraction for both the attacker and legitimate users.

Legitimate users may accidentally interact with honeyfiles, but even if they do, they are less likely to edit or transfer them. To capture this, we assign a *seriousness* level to the action taken on a honeyfile, to reflect the security threat it poses. We consider two levels of seriousness.

**Weak:** Open or close a honeyfile.

**Strong:** Edit, transfer, or apply tools such as `zip` and `tar`.

When anyone (e.g., the user or attacker) touches honeyfiles, the Mee client collects corresponding sensitivity and seriousness values and sends them to the Mee controller.

### Mee Controller

The Mee controller represents the defender across the network. It receives alarms from Mee clients and analyzes them to update its beliefs about the security level of each device. Then, it instructs each client to create or delete a specified number of honeyfiles on its device. Upon receiving an alarm, the controller determines whether the described access is by a legitimate user or an attacker. We define $HN_j$ as the number of honeyfiles on device $j$. This number is adjusted based on the defender's beliefs about how secure a device is and the attacker's goals. To assist the Mee Controller in making the optimal decision, we describe the following measures of the alarm analysis to assess network security situations.

**Device Group and Group Risk Level:** We assume that an insider attacker targets files placed on devices belonging to specific organizational roles, which we model as *groups*. For example, in a university, there may be groups for professors, students, and accountants. One device can be part of multiple groups; for example, one device may be in the "professor" ($G_{\text{professor}}$) and "engineering" ($G_{\text{engineering}}$) groups. Here, $G_i$ is the set of devices in group $i$, and $HG_i$ is the number of honeyfiles in group $i$, which is the same notation used for the number of honeyfiles in device $d$. The *Risk Level* of group $i$ is represented by $R_i$. A higher $R_i$ indicates a greater likelihood of being the target group of an attacker. If someone accesses a honeyfile on a device in group $i$, the defender increases the corresponding $R_i$.

**Group Risk Level Update:** Recall the sensitivity of a honeyfile and the seriousness of the action of Section 3.4. Equation 3.1 computes the change of risk level of group $g$ due to a single action $a$ on a specific honeyfile, $h$.

We define the group risk updating ($\Delta risk_g(h, a)$) as the product of the honeyfile sensitivity and the action seriousness divided by the number of honeyfiles in the group.

$$\triangle\text{risk}_g(h, a) = \frac{\text{sensitivity}_h * \text{seriousness}_a}{HG_g} \tag{3.1}$$

Through the group risk level update, the Mee controller evaluates the security situation for each group. To compare the security situation of a group with the rest of the network, we introduce $R_{-i}$ with a negative in the subscript to denote the average group risk level of all groups except the group $i$. Equation 3.2 shows how we calculate $R_{-i}$, where $NG$ denotes the number of groups in a network.

$$R_{-i} = \frac{\sum_{j \neq i} R_j}{NG - 1} \tag{3.2}$$

Based on $R_{-i}$ and $R_i$, we use the classification as *Dangerous*, *Medium*, and *Safe* to separate groups (as shown in Equation 3.3).

$$\textit{Classification} = \begin{cases} \text{Dangerous} & \text{if} \quad R_i > R_{-i} * 2 \\ \text{Medium} & \text{if} \quad R_{-i} < R_i < R_{-i} * 2 \\ \text{Safe} & \text{if} \quad R_i < R_{-i} \end{cases} \tag{3.3}$$

Via the group risk level, the Mee Controller can evaluate the security situation of all the groups. If the insider attacker tends to compromise the device based on the roles of the device owner, the Mee controller increases the number of honeyfiles in each device whose group obtains a high group risk level.

## Communication between Mee Client and Controller

As we discussed above, Mee clients and controller exchange messages by which the controller evaluates the network security situation and asks Mee clients to adjust the number of honeyfiles in the corresponding device. Therefore, we name *Honeyfile Alarm* as the messages that transfer from all Mee clients to the

Controller. Each honeyfile alarm contains a tuple $\langle CN, HN, AS \rangle$, where $CN$ represents the client name, $HN$ represents the honeyfile that is acted upon, and $AS$ represents the action on the triggered honeyfile. In contrast, *Command* is the message that comes from the Mee controller to each Mee client, which contains a tuple as $\langle CN, I \rangle$. Here, $CN$ represents the target host of the command, and $I$ represents the instructions that the Mee client needs to follow with.

The Mee system is shown in Figure 3.1. If anyone acts on a honeyfile, the Mee client sends the honeyfile alarm to the Mee controller. The alarm contains information, including the location of the triggered honeyfile, corresponding honeyfile sensitivity, and action seriousness, which are used to update the belief of the Mee Controller, such as each group's risk level. All alarms are stored in the Mee controller as well. Based on group risk and alarm history, the Mee controller sends the command to Mee clients to adjust the number of honeyfiles in the device or check the device if necessary.

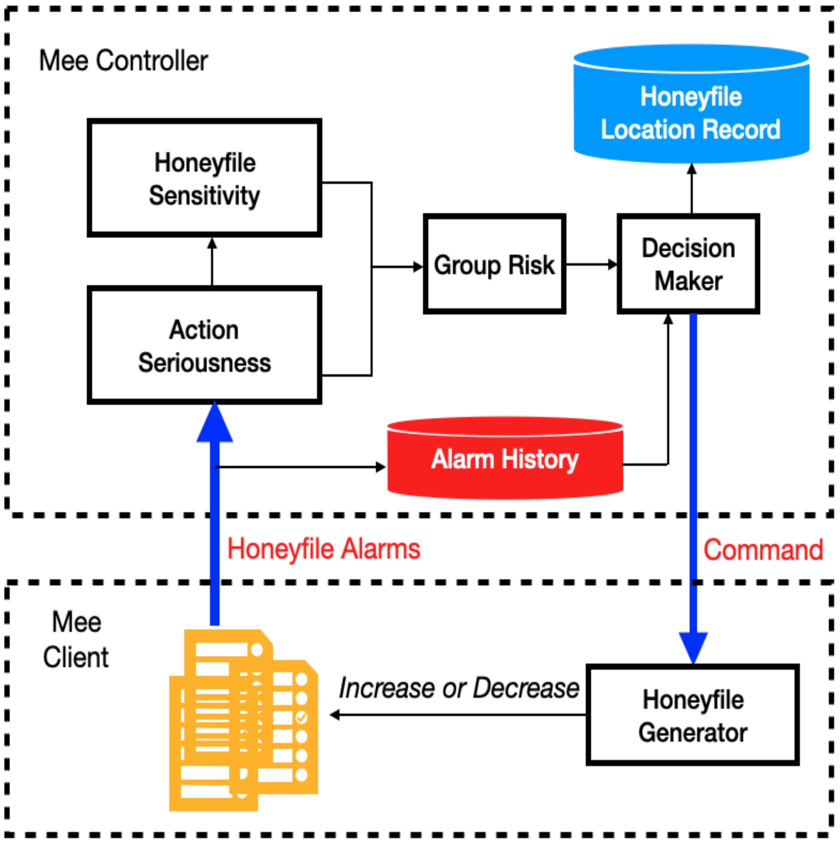


Figure 3.1: Mee System Structure

## 3.4.1 Model for Mee

In this section, we describe the model of insider attackers, the defender, and users, and also the interaction between them.

**Network and Node Model**

A network environment in an enterprise consists of various end-user devices, such as computers and laptops, that are connected to each other. These devices are categorized into different groups based on the roles of their respective owners. The security situation of each group is determined by its group risk level. Additionally, all personal devices that are connected, such as laptops, have a Mee client installed on them. This client has the capability to generate or delete honeyfiles. We represent the number of honeyfiles in device $i$ as $NF_i$. Each honeyfile created is assigned a value that signifies its attractiveness to both attackers and legitimate users, following the concept of file sensitivity.

**Attacker Model**

**Attacker Actions:** We consider the following action set:

**Penetrate Device ($A_i^1$):** Leverage legitimate authorization to penetrate a device (but without knowledge of its file system).

**Search ($A_i^2$):** Explores device $i$ to search for valuable files.

**File read ($A_i^3$):** Opens, reads, or closes a file on the device $i$.

**File Transfer or Modify ($A_i^4$):** Transfer, edit, or delete a file.

**Attacker Payoff:** We separate the attacker payoff into *Effectiveness (EA):* the benefit the attacker receives from an action; *Action Cost (AC):* cost to the attacker to deploy action $i$; and *Impact of Failure (IF):* cost to an attacker if the attacker touches a honeyfile.

**Defender Model**

Mee combines the decentralized deployment of honeyfiles with centralized control to adjust the number of honeyfiles in each connected device. Specifically, the Mee controller monitors the devices and decides how many honeyfiles to place on each device. The defender's goal is to maintain group risk levels and detect a compromised device.

**Defender Actions:** We design four actions for the defender.

**Check ($D_i^1$):** Inform the Mee client of a device to check for existing backdoors or update the OS and application to avoid vulnerabilities. Upon doing so, set the number of honeyfiles on the device resets to the initial value and the group risk level value to the $R_{-i}$ (defined in Section 3.4) value.

**Increase ($D_i^2$):** number of honeyfiles in the device $i$.

**Decrease ($D_i^3$):** number of honeyfiles in the device $i$.

**No change:** Maintain the current strategy and save resources.

Note that in Mee, the defender chooses an action based on alarms from deployed honeyfiles as well as the network security situation. For example, if the defender receives an alarm from a device in the dangerous group, the defender has a higher probability of choosing the Check action.

**Defender Payoff:** The defender's payoff has four parts.

**Effectiveness** ($ED$) Reward when a malicious action is detected, e.g. recovering the compromised device and misleading the attacker with honeyfile.

**Defense cost** ($DC$) Cost of deploying an action.

**Failure** to Protect Real File ($FR$): Punishment for failing to protect real files.

**Impact** on the legitimate user ($IN$): False positive if a user accidentally opens, closes, transfers, or modifies a honeyfile.

#### Model of Legitimate User

We model the behaviors of legitimate users to capture Mee's impact on them. Therefore, we design the user action set as:

**Login:** Access a device.

**Search:** Explore a device, for example, open a folder, and search for a file.

**Read** a file.

**Transfer, modify, or delete** a file.

No matter what their roles are, users can access their devices and read or modify their target files whenever they want. Consequently, there are two main distinctions between insider attackers and legitimate users in our threat and system models.

First, the login activities of legitimate users across the network are usually randomly distributed. Therefore, if no insider attacker is present, all devices have the same chance of being logged in by their owner. On the other hand, an insider attacker would exploit the device based on the roles of the device owner. Secondly, users can quickly locate a target file as they are familiar with the file system, but a masquerader, lacking such knowledge, would have a more random movement to search for valuable files. As a result, an insider attack has a higher probability of acting on a honeyfile than a legitimate user.

### 3.4.2 Honeyfile Game with Mee

We analyze the honeyfile game as a dynamic two-player Bayesian game in which the two players update their beliefs based on the progression of the game. Player $a$, the defender, deploys honeyfiles within connected devices to detect insider attackers. Player $b$ has two potential types: a legitimate user or an attacker. The defender's type is known to both players, but the defender does not know the type of the other player. The defender can form its belief by observing honeyfile alarms from each device. Let $t \in T = [0, 1]$ represent the type of player $b$, where $t = 0$ stands for a legitimate user and $t = 1$ for an attacker.

In the defender's belief, we use $\sigma_a^s = p$ to denote the probability that the player $b$ is an attacker ($t = 1$), and $\sigma_n^s = 1 - p$ to denote the probability that the player $b$ is a legitimate user ($t = 0$). An alarm from a Mee client represents an observation from player $b$. The defender has no knowledge of which device is compromised. Therefore, the honeyfile alarms may represent the detection of an attacker or false alarms from legitimate users. To make an optimal decision, the defender needs to estimate player

$b$'s type based on its belief, including the corresponding group risk level and the history of previous alarms. With the definition of action seriousness and the action sets of the two players, Figure 3.2 explains the decision tree between the two players.

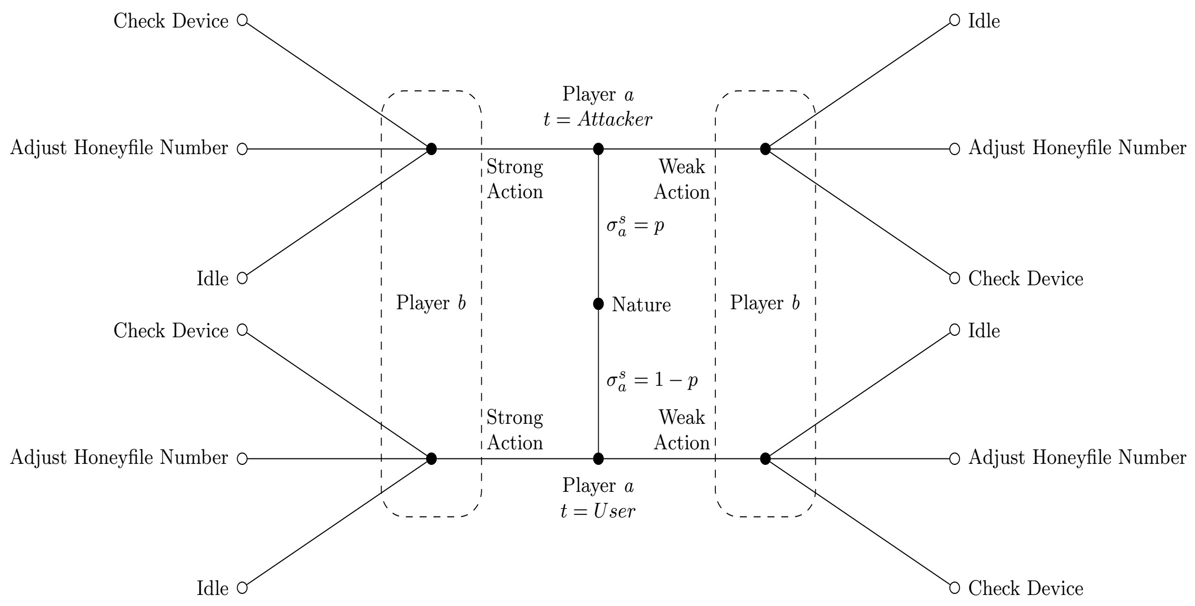


Figure 3.2: Decision tree of the honeyfile game: *Step 1:* player $a$ obtains a type from nature, which is the root of the decision tree, as its private information. *Step 2:* After player $a$ triggering a honeyfile alarm, if player $a$ is an attacker, the path of decision tree goes to the upside. On the other hand, the path goes down. *Step 3:* the defender (player $b$) chooses its action based on the perspective of player $a$'s type.

**Utility Function:** Let $w \in [1, 3]$ denote the worth of a regular file, where a higher number represents a more valuable file. For simplicity, we stipulate that an insider attacker obtains a gain of $w$ if it reads a file and obtains $2 * w$ if it transfers or modifies a file. Let $c_c$ denote the cost of compromising a device, $c_r$ represents the cost of reading a file, and $c_t$ represents the cost of transferring or modifying a file. Recall the definition of honeyfile sensitivity and action seriousness. We use $h_p$ to denote the punishment of an attacker when it acts on a honeyfile. For simplicity, we stipulate that the punishment for reading a honeyfile is $h_p$, and the punishment for transferring or modifying a honeyfile is $2 * h_p$. Let $\beta \in [0, 1]$ represent the probability that an insider attacker estimates an actual file. The attacker calculates $\beta$ based on its history, such as how many honeyfiles and actual files it touches.

We use $c_a$ to represent the cost when the defender adjusts (e.g., increases and decreases) the number of honeyfiles. Let $c_{cd}$ represent the cost of checking device action, and $r_{cd}$ represent the reward if the defender successfully recovers a compromised device. Meanwhile, $-r_{cd}$ represents the loss of an insider attacker if the defender recovers a compromised device. Let $\alpha$ represent the probability that the defender thinks the device is compromised.

Table 3.1 shows the expected payoff when an insider attacker acts on a file. Note that the payoff in

Table 3.1 does not include the cost of compromising a device. As a rational player, an insider attacker expects that the reward from real files in a device is larger than the cost of compromising the device. In other words, if $NR$ represents a set of real files that an attacker acts on (e.g., reads and transfers), it expects $c_c < \sum_{nr \in NR}(w_{nr} - c_{r(nr)}) + \sum_{nr \in NR}(2w_{nr} - c_{t(nr)})$.

Table 3.1: Player $a$ is an insider attacker. The tuples below include ⟨defender's payoff, attacker's payoff ⟩.

| | Read a file | Transfer or Modify a file |
|---|---|---|
| Check Device | $\alpha * r_{cd} - c_{cd}, \beta w - (1-\beta)h_p - r_{cd} - c_r$ | $\alpha * r_{cd} - c_{cd}, 2\beta w - (1-\beta)h_p - r_{cd} - c_t$ |
| Increase Honeyfile | $-c_a, \beta w - (1-\beta)h_f - c_r$ | $-c_a, 2\beta w - 2(1-\beta)h_f - c_t$ |
| Decrease Honeyfile | $-c_a, \beta w - (1-\beta)h_f - c_r$ | $-c_a, 2\beta w - 2(1-\beta)h_f - c_t$ |
| No Change | $0, \beta w - (1-\beta)h_f - c_r$ | $0, 2\beta w - 2(1-\beta)h_f - c_t$ |

Table 3.2: Player $a$ is a legitimate user. The tuples below include ⟨defender's cost, user's cost ⟩.

| | Read a file | Transfer or Modify a file |
|---|---|---|
| Check Device | $\alpha * r_{cd} - c_{cd}, 0$ | $\alpha * r_{cd} - c_{cd}, 0$ |
| Increase Honeyfile | $-c_a, 0$ | $-c_a, 0$ |
| Decrease Honeyfile | $-c_a, 0$ | $-c_a, 0$ |
| No Change | $0, 0$ | $0, 0$ |

**Dynamic Game:** The honeyfile game is played repeatedly between player $a$ and player $b$. Let $k_t$ represent the timeline of one game, where $t = 0, 1, 2, \ldots$. At any given time, only one player $b$ (e.g., an insider attacker or a legitimate user) is active. Player $b$ firstly obtains its type from nature as private information. When the player $b$ has achieved its goals, such as an attacker collecting enough valuable files or a user obtaining the target file, it exits the game and another player $b$ takes its place. Player $a$ (e.g., the defender) accumulates its beliefs throughout the game. Specifically, the defender updates its belief about the type of player $b$ after receiving honeyfile alarms. It chooses actions based on all the gathered information, such as group risk levels and honeyfile locations, from the start of the game.

### 3.4.3 Implementation and Evaluation for Mee

We compare Mee with the traditional honeyfile system via a simulation. In the testbed, we deploy 114 devices and separate them into 20 groups. The installed Mee client in each connected device can create or remove honeyfiles following the Mee controller's command. Meanwhile, the Mee controller can estimate the network security situation by receiving honeyfile alarms from Mee clients.

**Attacker Setting:** The action set of an insider attacker is {*Penetrate*, *Search*, *Read*, *Transfer*} (as introduced in Section 3.4.1). The insider attacker selects several groups as its *target groups* and has a higher probability of penetrating the devices within the target groups rather than randomly selecting a device to attack. Specifically, in all following tests, an attacker has a ten percent probability of randomly choosing a target device to compromise (without considering the target group) and a ninety percent probability of selecting a device in the target group(s).

Assume that an insider attacker can penetrate any device across the network and explore the compromised machine to search for valuable files. However, not every compromised device contains a valuable file. In each simulation, the attacker starts from an initial budget, representing the expected cost for searching and gathering valuable files on every device. Suppose the cost of searching for valuable files within a compromised device is more than the initial budget. The attacker can choose to abandon the current device and instead penetrate another device.

**User Setting:** To calculate the defender cost and false positive rate of the honeyfile alarm, we model the behaviors of legitimate users in our simulation. A user has an action set [*Login*, *Search*, *Read File*, *Transfer or Modify File*]. Additionally, the user gets a complete map of their file systems, which can assist the legitimate user access a target file faster than the attacker. However, every user has a 10% probability of choosing an incorrect action or action target. Thus, the user may act on a honeyfile and generate a false alarm.

Although the last three actions of the user model are the same as the corresponding actions in the attacker model, we emphasize the differences between legitimate users and the attacker. At first, each legitimate user has a clear target file and has complete knowledge of the file system. Thus, the user can locate any file on the device without a random search. And then, users access the corresponding devices at any time, so the distribution of login behaviors across the network is random. In contrast, the insider attacker tends to perform the compromise action relying on its target group(s) but randomly explores the file system.

**Defender Setting:** We simulate both the traditional honeyfile system, in which the number of honeyfiles in each device is static, and Mee, which dynamically adjusts the number of honeyfiles in each device. The defender has an action set [*Check Device*, *No Change*] for the traditional honeyfile system and an action set [*Check Device*, *No Change*, *increase honeyfile decrease honeyfile*] for Mee.

**Honeyfile Generation:** In accordance with Gómez-Hernández et al. [2018], we use `inotify-tools` to monitor honeyfiles. This library and set of command-line programs for Linux detect and respond to file system events, such as opening, closing, and modifying, on a honeyfile. The Mee client then sends this information to the Mee controller. For the purpose of this research, we only consider `txt` files. For the simulation of the traditional honeyfile system, the same number of honeyfiles is deployed in each device. We then change the number of honeyfiles in each device to observe the defender's performance, the attacker's cost, and the false/true positive rate. The defender can modify the number of honeyfiles depending on its assessment of the network security situation. For Mee, we initially deploy 40 honeyfiles in each device.

### Comparing Mee with the Traditional Honeyfile System

A traditional honeyfile system does not adjust the number of honeyfiles in each device and maintains the level of risk in the group. To simulate a traditional honeyfile system, we deploy a fixed number of honeyfiles in each connected device. We separate the devices into 20 groups and posit that the insider

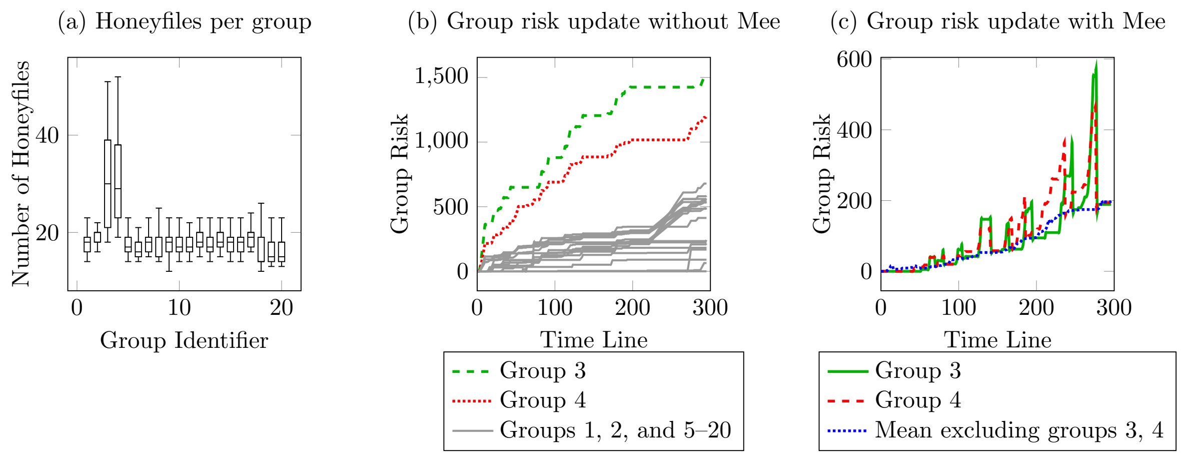


Figure 3.3: Mee's Performance: (a) Numbers of honeyfiles in different groups; (b) Group risk updates without Mee; (c) Group risk updates with Mee

attacker prefers to attack Group 3 (G3) and 4 (G4). We evaluate the performance of the traditional honeyfile system and Mee based on various numbers of honeyfiles and the group risk level update.

### Adjusting the Number of Honeyfiles with Mee

Mee changes the number of honeyfiles in each connected device to reduce unnecessary overhead and the impact on legitimate users. Figure 3.3(a) records the variance of the number of honeyfiles in each group. We assign ten insider attackers and 200 legitimate users in this test. Because the attacker's target groups are G3 and G4, Mee automatically adjusts and deploys more honeyfiles in these two groups than others. The average numbers of honeyfiles in G3 and G4 are 29 and 30, whereas the average number of honeyfiles in other groups is 18.

### Group Risk Level of the Traditional Honeyfile System and Mee

Figure 3.3(b) illustrates the results of the traditional honeyfile system's group risk level update. The X-axis displays the timeline of one test, and the Y-axis shows the group risk level. Each player takes part in the test and acts as either a legitimate user or an insider attacker in each time slot. The group risk levels of G3 and G4 (dotted red and dashed green lines) are much higher than the other groups (solid grey lines), indicating the tendency of the attacker's movement to be detected through the group risk level updating.

With the same setup, Mee controller receives and examines honeyfile alarms from Mee clients to maintain the group risk level. Figure 3.3(c) displays the group risk level update with Mee. Any individual who triggers the honeyfile alarm increases the group risk levels. Suppose Mee controller confirms that a group has an abnormal (e.g., much higher than other groups) group risk level. It can apply the *check* action and reset the corresponding group risk level to $R_{-i}$. In Figure 3.3(c), the red dashed line and solid green line represent the risk levels of the G3 and G4 groups (e.g. the target groups of the attacker). The blue dotted line is used for the rest of the group's risk levels to denote their average value.

### Defender Payoff with the Traditional Honeyfile System and with Mee

We calculate the defender's payoff by considering ten insider attackers and 200 legitimate users. We use the setting of Section 3.4.1 to calculate the players' costs. We increase the number of honeyfiles in each device from 0 to 100 for the traditional honeyfile system. Figure 3.4(a) shows that the defender payoff (blue dashed line) increases until the number of honeyfiles is less than forty, then decreases. Growing the number of honeyfiles can help the defender detect insider attackers more effectively, but too many honeyfiles can disrupt legitimate users and generate false positive alarms that confuse the defender. The orange line in the figure represents the defender payoff when Mee is deployed with the same attacker and user settings.

### Attacker Cost with Traditional Honeyfile System and Mee

With the same setting as above, Figure 3.4(b) records the attacker's payoff given a certain number of honeyfiles in the environment. The X-axis represents the number of honeyfiles in each device, and the Y-axis represents the attacker's payoff. With a traditional honeyfile system, more honeyfiles can significantly reduce the attacker's payoff. However, compared with Figure 3.4(a), the overhead of honeyfiles also disturbs the defender's performance. Also, the straight orange line represents the attacker payoff when we apply Mee.

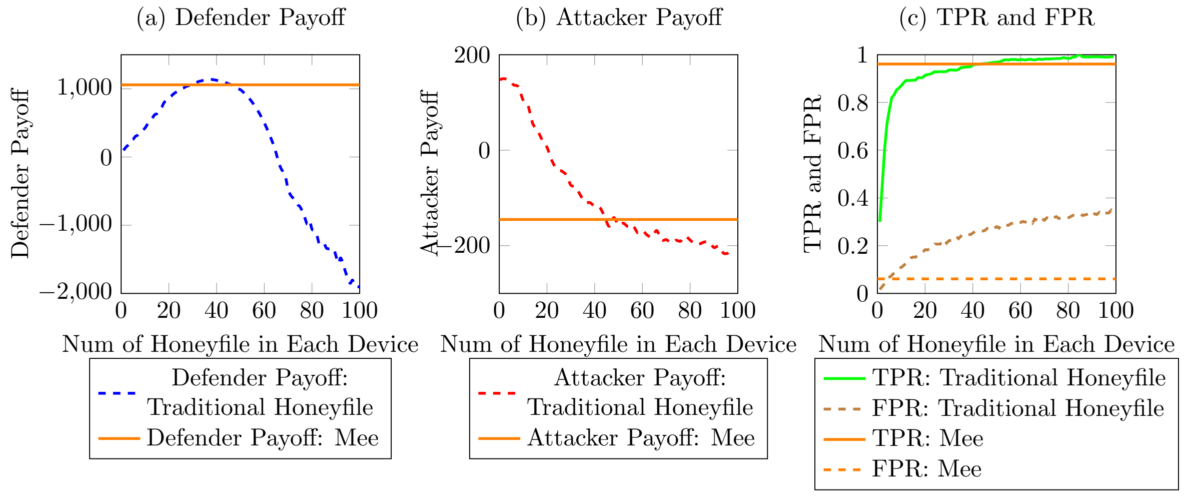


Figure 3.4: Comparison Between the Traditional Honeyfile System and Mee

### TPR and FPR with the Traditional Honeyfile System and with Mee

We calculate the false positive rate (FPR), true positive rate (TPR), and the area under the ROC curve for the traditional honeyfile system and Mee. The area under the ROC curve is defined as $ROCarea = TPR * (1 - FPR)$.

We consider positive, negative, true positive, and false positive as metrics for the defender.

**Positive:** Defender detects that an insider attacker is present.

**Negative:** Believes that the honeyfile alarm is triggered by a legitimate user.

**True positive:** Detection of the insider attacker rather than a legitimate user.

**False positive:** Misunderstanding of the honeyfile alarm triggered by a user.

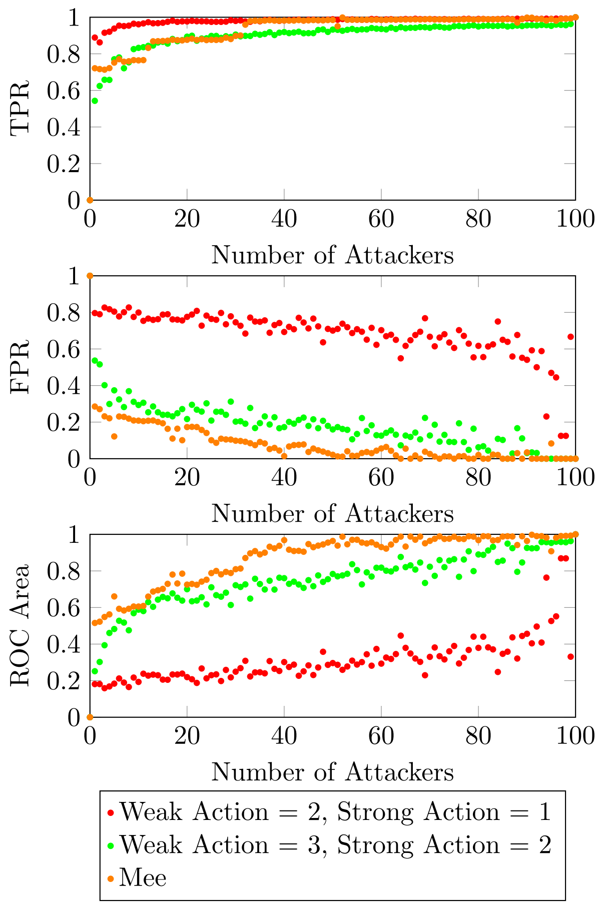


Figure 3.5: True and False Positive Rate of Traditional and Mee Honeyfile System

With the same test settings as above, Figure 3.4(c) shows the TPR and FPR of the traditional honeyfile system and Mee. The solid green line and dashed brown line show the TPR and FPT of the traditional honeyfile system, while the orange solid and dashed lines represent Mee. With the traditional honeyfile system, increasing the number of honeyfiles in each device can significantly raise the TPR but also bring a high FPR. Mee maintains the TPR at a high level and reduces the FPR.

We then evaluate the performance of the traditional honeyfile system and Mee with different numbers of insider attackers. According to the definition of action seriousness in Section 3.4, we define several scenarios for insider attacker detection with a traditional honeyfile system. Figure 3.5 shows the ROC results for different detection definitions. For instance, the red marks indicate that the defender is alerted if any action is taken on the honeyfile in one device with at least two weak actions or one strong action. We increase the number of insider attackers in each test from 1 to 100. As shown in Figure 3.5, when weak action=2 and strong action=1, the ROC value is concentrated in a high TPR and FPR range, indicating that detection is highly sensitive. However, a traditional approach can detect insider attackers but also generates many false positive alarms that can confuse the defender and affect legitimate users. Furthermore, increasing the number of attackers does not significantly affect the performance of the traditional honeyfile system. When weak action=3 and strong action=2 (shown as green marks in Figure 3.5), the FPR decreases but the TPR also decreases, which means that a large number of insider attackers can go undetected. The orange marks in Figure 3.5) represent the performance of Mee, which shows that Mee can reduce the FPR while maintaining a high TPR. The ROC area figure clearly demonstrates that Mee has a better performance.

## 3.5 HoneyMee

While Mee enhances the benefits of honeyfile deployment, the research environment of Mee is relatively simplistic. For example, the research only considers a single attacker or user in a time slot. Our objective is to create a more accurate representation of the environment and incorporate additional system details into our model.

In this section, we improve Mee by introducing a novel approach called HoneyMee. Rather than relying on game theory to help the defender choose the best solution, this approach utilizes deep reinforcement learning (DRL) to analyze the environment and find the optimal defensive strategy. HoneyMee shares a similar architecture with Mee, wherein it incorporates both centralized control (through the HoneyMee Controller) and decentralized deployment (through the HoneyMee Clients). Each connected device in HoneyMee is equipped with honeyfiles, and any alarms triggered by these honeyfiles are forwarded to a controller for further analysis. However, since HoneyMee utilizes DRL for information collection and analysis, the parameters and structure also require different configurations compared to Mee. Therefore, we redefine the structure and parameters as follows:

### 3.5.1 Structure

- **HoneyMee Client** is an endpoint application that is installed on all devices. It monitors honeyfiles on the device and detects any suspicious activity, such as opening, editing, deleting, or transferring a honeyfile. Whenever a honeyfile is accessed, the HoneyMee Client sends an alert to the HoneyMee Controller. Meanwhile, the HoneyMee Client waits for the instruction from the HoneyMee Controller after sending the alert. We assign $HN_j$ as the number of honeyfiles on the device $j$.

- **HoneyMee Controller** represents the defender of the network. It receives alarms from HoneyMee Clients and analyzes them to update its beliefs about the security level of each device. Upon receiving an alarm, the controller determines whether the described access is by a legitimate user or an attacker based on its understanding of the whole environment situation. We have created a DRL model to assist the HoneyMee Controller in obtaining the most advantageous strategy, considering users, attackers, and defenders.

- **Communication between HoneyMee clients and the Controller** includes all exchanged messages. We see *Honeyfile Alarm* as the messages transferred from all HoneyMee Clients to the HoneyMee Controller. Each honeyfile alarm contains a tuple $\langle CN, HN, AS \rangle$, where $CN$ represents the client name, $HN$ represents the honeyfile that is acted on, and $AS$ represents the action on the triggered honeyfile.

### 3.5.2 Parameters

We assign several parameters to help the defender analyze each honeyfile alarm and assess the network security situation.

**User's Active Time:** The HoneyMee Controller collects data on the amount of time each device is active to identify the user's usage habits. For example, one user may be more likely to use their device during the day rather than in the evening. This implies that if a honeyfile is touched during the day, it is more likely to be a mis-interaction by the user. However, if a honeyfile is triggered at night, there is a higher probability that the alert is caused by an insider attacker. On the other hand, a device with a longer active time may experience more frequent interactions with the honeyfile than a device with a shorter active time.

Let $UA_i$ denotes the log of active time record for device $i$.

$$UA_i = \{A_1 : [t_1, t_2], A_2 : [t_3, t_4] \ldots A_n : [t_n, t_m]\}, \tag{3.4}$$

where $A_n$ is the duration of active time, with the first item of the list being the time of user login and the second item being the time of user logout. The HoneyMee Controller gathers all $UA_i$ and creates a dynamic probability matrix to show the likelihood of each device being active at any given time slot.

**Honeyfile Sensitivity:** The honeyfile sensitivity, $HS \in [1, 3]$, indicates how attractive the honeyfile is to both attackers and legitimate users. As Yuill et al. [2004] and Salem and Stolfo [2011] have noted, the name and location of a honeyfile can be a major draw for attackers. For example, a honeyfile with the name `2023-NLP-Final-Exam.pdf` located in a folder called "Exam" may be especially appealing to an attacker looking to steal the exam paper. However, this kind of naming and placement could also lead to more confusion for the legitimate user, and increase the likelihood that the well-crafted honeyfile will be mistakenly accessed by the authorized user. Thus, the HoneyMee Controller will take into account the sensitivity of the honeyfile if it receives alarms from HoneyMee Clients.

**Action Seriousness:** The severity of the action taken, denoted by $AC \in [1, 2]$, reflects the potential danger posed by the action taken on the honeyfile [Salem and Stolfo, 2011; Yuill et al., 2004]. We distinguish between two levels of severity: minor action (opening or closing a honeyfile) and major action (transferring, editing, or using tools such as `zip` and `tar`).

Legitimate users may unintentionally interact with honeyfiles. Even if they open or view one, there is a small chance that they will try to alter or move the honeyfiles. On the other hand, an insider attacker has a greater likelihood of taking major actions on a honeyfile than a user. Based on this assumption, action seriousness is considered to calculate the system security situation when a honeyfile alarm is triggered.

**Organization Roles:** Recall the definition of *organization roles* (introduced in Section 3.3). We assume that each host in the environment has organizational roles, such as a professor or a student.

An insider attacker targets files placed on devices belonging to specific organizational roles, which we model as *groups*. We denote the set of devices in the group $i$ as $G_i$. A single device can be part of multiple groups. For example, in a university, there may be groups for professors, students, and several apartments. One device may belong to both the "professor" ($G_{\text{professor}}$) and "CSC apartment" ($G_{\text{CSC}}$) groups.

To differentiate the groups, we assign an *Importance* value ($I \in [1, 5]$) to each group according to how important it is to the defender. The higher the importance value, the more valuable the group is for the defender and also more attractive to attackers. We use $GS_i$ to represent the *group security values* of group $i$. A higher $GS_i$ indicates a higher probability that an attacker targets the group. If someone touches a device's honeyfile in group $i$, the defender increases the corresponding $GS_i$. We assign $HG_i = \sum_{j\in i} HN_j$ as the number of honeyfiles in the group $i$, reusing the notation for the number of honeyfiles in the device $j$ ($HN_j$). Taking into account the sensitivity of the honeyfile, the seriousness of the action, and the importance of the group, Equation 3.5 computes the change in the risk level of the group $g$ due to a single action $a$ on a specific honeyfile, $h$. We define the group risk update ($\Delta GS_g(h, a)$) as the product of the sensitivity of the honeyfile and the seriousness of the action divided by the number of honeyfiles in the group.

$$\triangle \text{GS}_g(h, a) = \frac{\text{sensitivity}_h * \text{seriousness}_a * \text{importance}_g}{HG_g} \tag{3.5}$$

Through the group risk level update, the HoneyMee Controller evaluates the security situation for each group. To compare the security situation of a group with the rest of the network, we introduce $R_{-i}$ (with a negative in the subscript) to denote the average risk level of the group of all groups except group $i$. Equation 3.6 shows how we calculate $R_{-i}$, where $NG$ denotes the number of groups in a network.

$$R_{-i} = \frac{\sum_{j\neq i} R_j}{NG - 1} \tag{3.6}$$

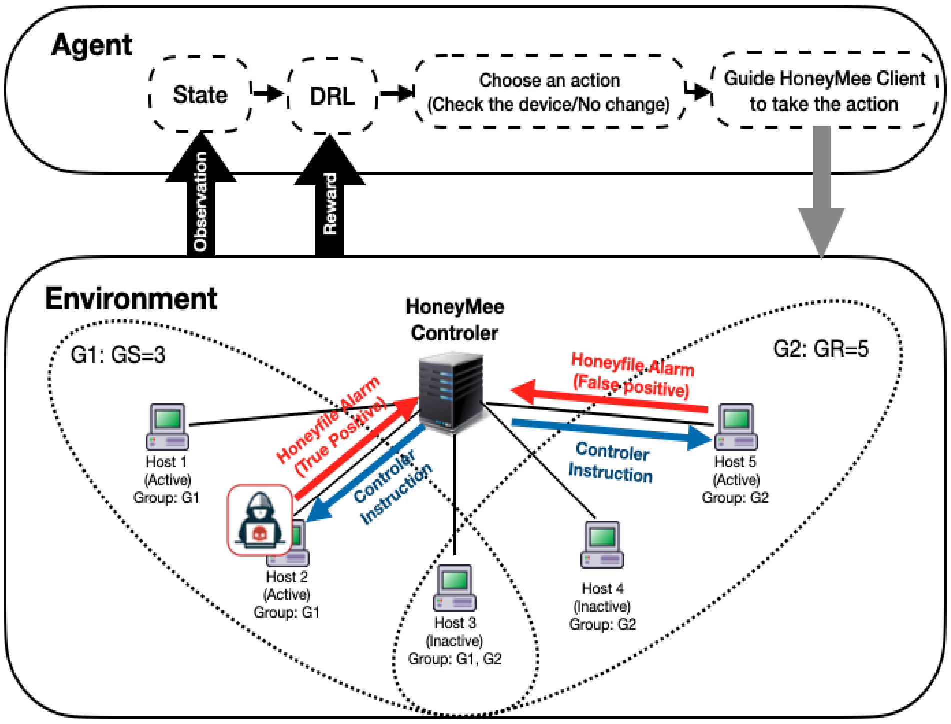
$R_{-i}$ is used to show if group $i$'s security situation is worse than other groups.

Table 3.3: Summary of notation

| Symbol | Meaning | Range |
|---|---|---|
| | **Environment** | |
| $NF_i$ | Number of honeyfiles in device $i$ | None |
| $HS_h$ | Honeyfile sensitiveness of honeyfile $h$ | [1,3] |
| $AC_a$ | Action seriousness of users or attackers' action $a$ | [1,3] |
| $SW$ | strong warning when someone triggers a honeyfile alarm | None |
| | **Attacker** | |
| $A1_i$ | attacker explores the device $i$ | None |
| | **Defender** | |
| $DE$ | reward of the defender when it correctly recovers a compromised device or misleads attacker by honeyfile | [1,10] |
| DC | cost of each defender's action | [1,5] |
| $DF_i$ | impact if the defender fails to protect the real file | $AE_i$ |

### 3.5.3 Deep Reinforcement Learning

We apply DRL to help the defender identify which honeyfile alarms indicate insider threats. In DRL, an agent interacts with the environment ($E$) by taking actions ($a \in A$) and observing states ($s \in S$). The agent searches for the policy $\pi(a_t s_t)$ that will maximize the expected payoff according to a reward function $r_t(s, a)$ at time $t$. This section explains the agent, state, and environment and how DRL can be used to find the optimal policy.



Agent
State
DRL
Choose an action (Check the device/No change)
Guide HoneyMee Client to take the action
Observation
Reward
Environment
HoneyMee Controler
G1: GS=3
G2: GR=5
Honeyfile Alarm (True Positive)
Controler Instruction
Honeyfile Alarm (False positive)
Controler Instruction
Host 1 (Active) Group: G1
Host 2 (Active) Group: G1
Host 3 (Inactive) Group: G1, G2
Host 4 (Inactive) Group: G2
Host 5 (Active) Group: G2

Figure 3.6: The HoneyMee Structure encompasses all connected devices, legitimate users, and potential insider attackers. If any interaction with a honeyfile occurs, the HoneyMee Client sends an alarm to the HoneyMee Controller, which the DRL agent observes. The DRL agent then selects the optimal strategy based on the current state, aiming to achieve the highest reward.

#### Environment

Connected devices are divided into groups based on the roles of their owners. Each group's security level is indicated by its security value. Each personal device (e.g., a laptop) has the HoneyMee Client installed, which monitors honeyfiles and sends notifications to the HoneyMee Controller. Let $HN_i$ be the number of honeyfiles in device $i$. Each honeyfile is assigned a value to reflect its appeal to attackers and legitimate users based on the sensitivity of the file. Furthermore, all active users and attackers are included in the DRL environment.

**Insider Attacker:** The range of activities available to an insider attacker includes penetrating, searching, reading, and transferring. An insider attacker is more likely to successfully infiltrate devices within certain target groups than randomly selecting a device to attack. To simplify the attacker model, in all tests, the probability of randomly choosing a target device to compromise (without considering the target group) is 10%, while the probability of selecting a device in the target group (s) is 90%. After the attackers penetrate any device on the network, they can explore the compromised machine to search for valuable files. However, not every compromised device contains a valuable file. The attacker has an initial budget ($IB$) to cover the cost of searching and collecting valuable files on each device. If the cost of searching for valuable files within a compromised device is greater than the initial budget, the attacker can choose to abandon the current device and instead penetrate another device. Attackers can infiltrate their target devices at any time.

**Legitimate User:** Users can perform various actions, such as logging in, searching, reading a file, and transferring or modifying a file. Legitimate users usually log in to their devices following their work and lifestyle patterns, such as those with a greater likelihood of activity during working hours, and some may be more likely to be active during the night. If a user logs in the device, the corresponding device is marked as *active* ($A$) (as discussed in Sec. 3.5.2). Unlike attackers, users have access to a comprehensive map of their file systems that can assist them in locating the needed file more quickly and accurately than an attacker. However, users may make a mistake and select the wrong action or target, resulting in them acting on a honeyfile and setting off a false alarm.

## Agent

HoneyMee Controller is the agent responsible for the reinforcement learning scheme.

**Action Space:** The actions for the controller are [*Check Device*, *No Change*]. Once the HoneyMee Controller is notified by a HoneyMee Client, it must decide on an action based on its understanding of the environment. The **check device** action suggests that the defender believes the device is being targeted and needs to be checked, such as prompting the user to alter passwords or upgrade the device's operating system to avoid any potential security issues. The **no change** action implies that the defender believes that the honeyfile alert is a false positive or that further observation is required before making a decision.

We define $A_D = \{a_1, a_2 \ldots a_t\}$ as the defender action set, where $a_t$ is the defender action at time $t$. The agent selects an action $a_t$ and receives a reward $r_t$ for each time slot. The agent's observations include alarms of honeyfiles and group security values, which may indicate the presence of an insider attacker or a legitimate user's mistake. The agent accumulates its beliefs and looks for a policy to help it choose the optimal action.

**State:** The matrix of the agent state ($S_t$) at time $t$ contains data on all devices, such as whether they are active ($a_i \in [0, 1]$) and the corresponding group security values ($GS_j$).

$$S_t = \Big\{ \ h_1 : [a_i, GS_j], h_2 : [a_i, GS_k] \ldots h_n : [a_i, GS_x] \ \Big\} \ , \tag{3.7}$$

where $h_n$ represents the device name.

**Reward Function:** The agent's reward function considers the following four parts.

- Effectiveness (ED): Reward when an insider attacker is detected.
- Defence cost (DC): Cost of deploying an action.
- Failure in protecting real files (FR): Punishment upon failing to protect real files, e.g., the insider attacker successfully touches valuable files and escapes.
- Wrong detection (WD): False positive detection after a user accidentally touched a honeyfile.

After an action $a_t$, the agent receives an immediate reward $r_t$. The agent aims to interact with the environment by selecting an action and maximizing its future rewards.

$$r_t = \begin{cases} -AC_a * HS_h & (\text{Attacker=True}, a_t = \text{No Change}) \\ ED - DC & (\text{Attacker=True}, a_t = \text{Check}) \\ WD & (\text{Attacker=False}, a_t = \text{Check}) \\ 0 & (\text{Attacker=False}, a_t = \text{No Change}) \end{cases} \tag{3.8}$$

### 3.5.4 Implementation and Simulation

We assess the effectiveness of HoneyMee via experiments in a simulated environment. The timeline of each episode is designed to span a week, with seven days in total. To simplify things, each day is divided into 100 time slots in which attackers or users can join, select actions, or quit the processes at any time slot.

#### Attacker Setting

We simulate a situation in which two attackers can be present in the environment at any given time. The attacker's action space is {*Penetrate*, *Search File*, *Read File*, *Modify File*}. An insider attacker is more likely to penetrate devices within certain groups than to select a device to attack randomly. In all tests, the probability of randomly choosing a target device to compromise (without considering the target group) is 10%, while the probability of selecting a device in the target group is 90%.

An insider attacker may be able to access any device on the network and search the compromised device for valuable files. However, not all compromised devices contain a valuable file. In each simulation, the attacker begins with an initial budget, which reflects the expected cost of searching and obtaining the valuable files on each device. If the cost of searching for valuable files on a compromised device is more than the initial budget, the attacker can choose to abandon the current device and penetrate another one. Here, we use DRL only for the defender and a greedy algorithm for the attacker.

#### User Setting

To calculate the cost and false positive rate of the honeyfile alarm for the defender, we simulate the behaviors of legitimate users. We conducted an analysis of 114 devices, each of which had 10–20 directories. Each directory contained 5–20 documents and 1–3 honeyfiles. As mentioned in Section 3.5.2, the honeyfiles have varying levels of sensitivity and are attractive to both attackers and users.

The action set for a legitimate user includes {*Login*, *Search*, *Read File*, *Modify File*, *Logout*}. Additionally, the user has a complete file system map, which helps them access a target file more quickly than an attacker. However, there is a 10% chance that the user will choose an incorrect action or action target, which could trigger a false alarm. The active time of a user when they log in to the device is relevant in analyzing honeyfile alarms across the network. Although the last three actions of the user model are the same as the corresponding actions in the attacker model, there are key differences between legitimate users and the attacker. Legitimate users have a clear target file and complete knowledge of the file system, so they can locate any file on the device without a random search. In addition, users can access the corresponding devices at any time, making the distribution of login behaviors across the network random. In contrast, the insider attacker tends to perform the compromise action relying on its target groups but randomly explores the file system.

Table 3.4: Models Setting and Test Results

| Model Name | Dense Layer | Batch Size | Max Reward | Min Reward | Average Reward |
|---|---|---|---|---|---|
| M0 | [32, 32] | 16 | 1355 | –515 | 299 |
| M1 | [32,32] | 32 | 1370 | –150 | 806 |
| M2 | [32,32,16] | 32 | 1990 | 1000 | 1552.25 |
| M3 | [32,32,32] | 32 | 1250 | –50 | –169.85 |
| **M4** | **[128,64,32]** | **32** | **2935** | **1230** | **2283.45** |
| M5 | [128,64,32,8] | 32 | 2820 | 1775 | 2176.5 |

### DRL Setting

We investigate various settings of DRL by testing different values to construct the neural network, such as the number of dense layers and units. We compare the maximum reward obtained in an episode, the minimum reward obtained in an episode, and the average reward after training. Table 3.4 presents the results of our experiments, based on which, we constructed a neural network with three dense layers and a batch size of 32, and the number of units in each layer being $[128, 64, 32]$, respectively. The graph in Figure 3.7 shows the TPR, FPR, FNR, and TNR of HoneyMee when it is trained with both legitimate users and insider attackers. Initially, these rates were close to 0.5. After 100 training episodes, the TPR (green line) and TNR (orange line) increased to almost 0.9, while the FNR (blue line) and FPR (red line) decreased to less than 0.2. We then used this trained model to compare with other models with different settings in the following test.

### Comparing Scheme

To demonstrate the effectiveness of the HoneyMee system in enhancing network security, we conducted a comparative analysis with various algorithms and techniques. We set up the simulation and generate results of HoneyMee ($H$) performance. We conduct a simulation in which the defender exhibits a greedy behavior ($G$) with the honeyfile alarm. This implies that the defender will identify anyone who engages with the honeyfile as an attacker. We then simulate the defender in a honeyfile environment where the defender selects action randomly ($R$). Insider attackers can be detected by the defender through the use of intrusion detection system ($IDS$) [Liu et al., 2007; Wahab et al., 2017]. To further evaluate the effectiveness of this approach, we conduct a comparison with the HoneyMee method using identical simulation parameters.

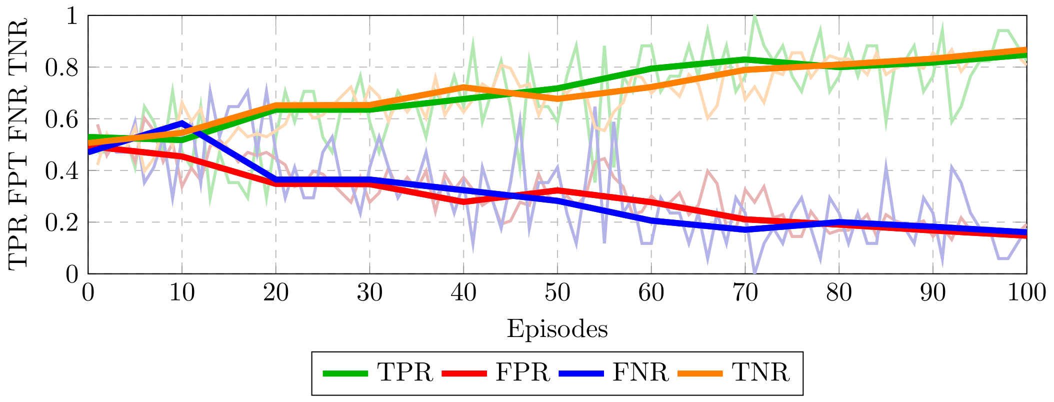


Figure 3.7: True and False Positive Rates of HoneyMee Traning Processes

## Results

**Defender and Attacker's Rewards** We initially conduct simulations of the techniques within an environment comprising 114 hosts and 3 insider attackers. Users interact with their devices by logging in and out based on their daily routines. The attackers infiltrate the target devices at various random time intervals. The parameters in Section 3.5.3 determine the defender's payoff, which is represented by $\langle \mathrm{AC}_a, \mathrm{HS}_h, \mathrm{ED}, \mathrm{DC}, \mathrm{WD} \rangle$.

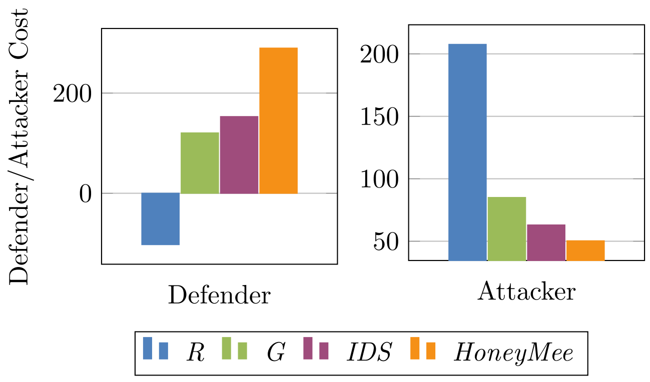


Figure 3.8: Comparation of Defender Cost and Attacker Cost

Figure 3.8 illustrates the comparison of rewards between the defender and attackers when the defender deploys various defense strategies. The mean defender and attacker's reward for Random action selection ($R$) are -102.444 and 207.57, indicating that while $R$ is capable of identifying attackers, it also significantly disrupts legitimate users. We consider $R$ as the baseline and the starting point for comparison. The utilization of greedy action selection ($G$) can assist the defender in identifying potential attackers, with the average rewards for the defender and attackers being 120.435 and 85, respectively. IDS, like $G$, has the ability to increase the defender's payoff to 153 and decrease the attacker's payoff to 63. Nonetheless, the presence of false positives still affects the defender's effectiveness. HoneyMee assists the defender in achieving a payoff of 289.7, while the attacker's payoff is 50.3. HoneyMee improves the defender's payoff by enhancing the identification of insider attackers more efficiently than alternative approaches, while causing minimal interference to legitimate users.

**TPR and FPR** To compare the accuracy of HoneyMee with other approaches, we define positive and negative outcomes as below.

- Positive: Detects that an insider attacker is present.
- Negative: Believes that the honeyfile alarm is triggered by a legitimate user.

On the basis of these, we calculate the false positive rate (FPR) and true positive rate (TPR).

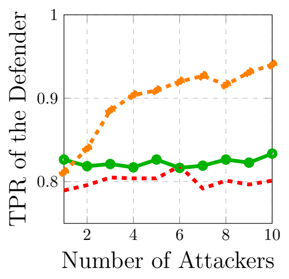
- True positive (TP): Identifying an insider attacker instead of a legitimate user.
- False positive (FP): Misinterpreting the honeyfile alarm activated by a user.
- True negative (TN): The system accurately categorizes a legitimate activity as negative or normal.
- False negative (FN): Cases where the system does not recognize a real attack.

The TPR and FPR are defined as follows:

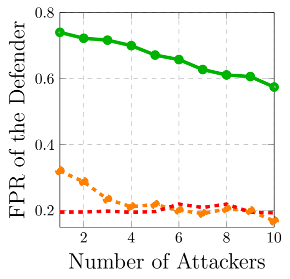
$$TPR = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}; FPR = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{3.9}$$

To assess the efficacy of HoneyMee and other approaches, various environmental settings are employed, including variations in the number of attackers and connected hosts. Initially, the quantity of insider attackers is raised from 1 to 10 while maintaining the number of connected devices at 114. Following this, the TPR and FPR of Greedy ($G$), IDS, and HoneyMee are calculated and compared for further examination. As shown in Fig. 3.9a, the *IDS* technique exhibits consistent TPR values, with *IDS* showing TPR values ranging between 0.79 and 0.82. The TPR for $G$ fluctuates between 0.81 and 0.83 due to the defender treating any interaction with the honeyfile as an attack, resulting in a minimal impact on the TPR value of $G$ even if the number of attackers increases. The TPR of *HoneyMee* increases from 0.81 to 0.94 with the growing number of attackers. This increase is attributed to the higher likelihood that the defender will detect an attacker rather than a legitimate user when the honeyfile alarm is activated, leading to an elevated TPR value for Honeyfile with the increasing number of attackers. Fig. 3.9b represents the FPR of the defender under various defense strategies. *IDS* shows consistent performance with an FPR ranging between 0.19 and 0.21. The FPR values of $G$ and *HoneyMee* decrease as the number of attackers increases, as a greater number of attackers leads to fewer false positive detections. However, the FPR of *HoneyMee* fluctuates between 0.32 and 0.16, while the FPR of $G$ decreases from 0.74 to 0.57.

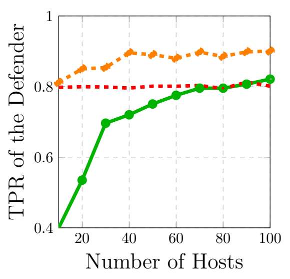
Similarity, Fig 3.9c and 3.9d illustrate the TPR and FPR results for $G$, $IDS$, and *HoneyMee*, where the number of attackers remains constant at 3 in every trial, while the number of hosts ranges from 10 to 100. As the ratio of attackers to hosts decreases, the TPR and FPR values of $G$ and *HoneyMee* decrease, while the stability of the *IDS* remains unchanged. The findings illustrated in Fig 3.9 suggest that a higher number of attackers and hosts can lead to improved performance of HoneyMee in comparison to other techniques.



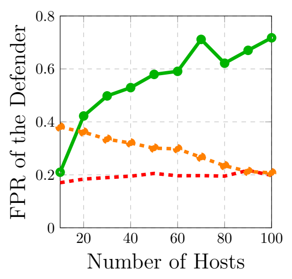
(a) TPR of the Defender as the Number of Attackers Increases



(b) FPR of the Defender as the Number of Attackers Increases



(c) TPR of the Defender as the Number of Hosts Increases



(d) FPR of the Defender as the Number of Hosts Increases

Figure 3.9: Comparison of True Positive Rate (TPR) and False Positive Rate (FPR) under Various Environment Configurations

## 3.6 Conclusion

Defensive deception technologies are commonly utilized for the purpose of delaying and identifying stealth attacks. Honeyfile systems are a straightforward and lightweight form of deception technology that sees widespread application. We introduce Mee and HoneyMee as innovative honeyfile systems intended to confuse and identify insider attackers. These two approaches make use of a *centralized* control and *decentralized* deployment to receive and analyze honeyfile alarms from connected devices. The objectives of this research include aiding the defender in minimizing the overhead and false positive detections arising from legitimate users.

In the Mee approach, Bayesian game theory is employed to model the interactions among the defender, the attacker, and legitimate users. To evaluate the performance of Mee, we conduct simulations and compare it with the conventional honeyfile system. The findings demonstrate that Mee can significantly decrease the average number of honeyfiles across the entire network. Consequently, Mee avoids the need for deploying unnecessary honeyfiles and mitigates the impact on legitimate users. One limitation of this approach is that the game model can only address relatively simple scenarios and environments, providing limited insights into the roles of the defender, attackers, and users. To address these gaps, we expand the Mee study by introducing a DRL-based honeyfile system named HoneyMee.

HoneyMee is a decentralized deployment and centralized control honeyfile system designed to aid defenders in detecting insider attackers. Our objective is to support defenders in distinguishing between true positive alarms and those triggered by legitimate users. To achieve this, an defender must select alarms to shape their perception of the system's security. We propose a DRL scheme to assist the defender in identifying the optimal solution. The implementation of HoneyMee with DRL in a simulated system treats all users and insider attackers as components of the environment. The outcomes reveal that HoneyMee exhibits a higher true positive rate and a lower false positive rate compared to other defense strategies such as IDS and greedy action selection. The results suggest that HoneyMee effectively enhances the detection rate in the presence of attackers and reduces the impact on legitimate users who unintentionally interact with honeyfiles. Additionally, we calculate the payoff for the defender and the insider attackers, indicating that HoneyMee can assist the defender in significantly reducing the costs associated with maintaining security while increasing the costs for attackers attempting to access target files.

CHAPTER

# 4 HONEYPOT-BASED CYBERDECEPTION AGAINST MALICIOUS RECONNAISSANCE

Malicious reconnaissance is a critical step for attackers in collecting enough network knowledge and choosing valuable targets for intrusion. Defensive deception (DD) is an essential strategy against threats by misleading attackers' observations and beliefs. Honeypots are widely used for cyberdeception that aims to confuse attackers and waste their resources and efforts. Defenders may use low-interaction honeypots or high-interaction honeypots. In this chapter, we consider a hybrid honeypot system that balances the use of the two levels of honeypot complexity, where high-interaction honeypots are more capable of deceiving skilled attackers than low-interaction honeypots. We present a two-player hypergame model that characterizes how a defender should deploy low and high-interaction honeypots to defend the network against malicious reconnaissance activities. We model the tradeoff of each player and characterize their best strategies within a hypergame framework that considers the imperfect knowledge of each player toward their opponent. Finally, our numerical results validate the effectiveness of the proposed honeypot system.

## 4.1 Introduction

Reconnaissance is a critical step that attackers perform to identify vulnerable and valuable targets. We consider *passive monitoring* and *active probing* of an enterprise network [Bartlett et al., 2007]. The passive monitoring attacks search services by monitoring the traffic between servers and clients and aims to use the collected intelligence to perform future attacks. Thus, they are often invisible (i.e., hard to be detected by the system) but consume fewer resources to launch the attacks. On the other hand, active probing attacks aim to identify services by aggressively sending packets to hosts and analyzing their responses. They finally identify vulnerable nodes and their criticalities in a target system. Although active probing is more effective in terms of actual attack benefits, it consumes more resources and can be easily detected by the system due to its active scanning.

To detect and mitigate malicious reconnaissance, we consider defensive deception using honeypots in this work. Honeypots are commonly used to detect and mislead attackers and are mainly categorized into low-interaction honeypots (LHs) and high-interaction honeypots (HHs). LHs behave in a more-or-less fixed way while HHs carry out more realistic interactions [Han et al., 2018]. LHs are easier to build and operate than HHs, while attackers can easily detect them. Although the attacker may not know an entire network topology or system information (e.g., active nodes' IP addresses), it may be able to distinguish the LHs from actual nodes by performing active probing.

In this work, we are interested in considering how the target system (i.e., a defender) can strategically identify an optimal defensive deception strategy (e.g., LHs or HHs) against such malicious reconnaissance where it aims to best protect the system from such attacks, given limited resources. Given resource constraints, we also would like to consider intelligent attackers who can strategically perform an optimal reconnaissance attack (e.g., passive or active scanning). Further, the attacker and the defender may perceive the game and their corresponding opponent's move differently under inherent uncertainty due to their partial observability. To effectively deal with such uncertainties in a game setting, we consider the so-called *hypergame theory* which enables players to choose their best action based on hypergame expected utilities (HEUs). The HEUs estimate players' expected utilities by considering uncertainty introduced by imperfect, partial observations of the game. To be specific, we make the following **key contributions** in this work:

- We develop a defensive deception framework based on a two-player hypergame for a setting in which an attacker applies active probing and passive monitoring while the defender deploys a mix of LHs and HHs to detect an attack. No prior work has studied game-theoretic defensive deception solutions for the attacker's game-theoretic strategic scanning attacks (i.e., passive or active).
- We identify the optimal defensive deception strategies considering each party's perceived uncertainty and hypergame expected utility (HEU). The two-player hypergame deals with uncertainty where the attacker and defender take actions based on their subjective perception of the game and the opponent's move.
- Via extensive simulation experiments, we prove that our game model can provide an optimal defensive deception strategy to effectively defend against reconnaissance attacks.

### Structure of This Chapter

This chapter is organized in the following way: Section 4.4 introduces the system model of this study. Section 4.5 examines the hypergame model which explains the interaction between attackers and the defender. Section 4.6 is the implementation of the hypergame-based hybrid honeypot system and the test results. Finally, Section 4.7 summarizes the chapter.

## 4.2 Related Work

Recent research has explored various methods to disguise network traffic, such as packet morphing, sending dummy traffic, and adding false information with packet features [Dyer et al., 2015; Pinheiro et al., 2018]. Anjum et al. [2020] proposed the use of fake flows to expose fake vulnerabilities in a computer network and delay advanced attacks. They suggested optimizing the generation of fake flows by network administrators to display false information and prevent attackers from gathering information. However, creating decoy flows that are indistinguishable from real flows is difficult. Adversarial machine learning techniques are being used to conceal network traffic and mask the fingerprints of packet features, where adding minor perturbations to the inputs can make classification more difficult [Granados et al., 2020].

The use of game theory in cybersecurity research has been widely discussed Zhu et al. [2021b]. Schlenker et al. [2020] proposed a deception game in which a defender chooses a deceptive response to an attacker's observation, with the attacker being either unaware or aware of the deception. Pawlick and Zhu [2015] developed a honeypot-based defense system using a signaling game, allowing an attacker to detect honeypots. However, previous work [Pawlick and Zhu, 2015; Schlenker et al., 2020] did not take into account honeypot systems with high- and low-value honeypots when uncertainty in expected utilities is considered using hypergame theory.

Hypergame theory [Fraser and Hipel, 1984] has been used as a comprehensive game model to represent different subjective perspectives between players in uncertain conditions. Vane and Lehner [1999] study hypergames for decision making in competitive contexts. Ferguson-Walter et al. [2019a] use hypergames to measure how a defensive deception signal can influence an attacker's beliefs. Cho et al. [2019] and Wan et al. [2021] discussed hypergame-based deception techniques for advanced persistent threat (APT) attacks that involve multiple stages of the cyber kill chain. Kulkarni et al. [2020] developed a zero-sum hypergame of incomplete information to evaluate the effectiveness of a proposed honeypot allocation. Unlike previous work [Cho et al., 2019; Ferguson-Walter et al., 2019a; Kulkarni et al., 2020; Wan et al., 2021], our work mainly focuses on constructing a defensive deception game framework in which the defender employs two types of honeypots (i.e., HHs and LHs) while the attacker performs either passive or active reconnaissance attacks.

## 4.3 Problem Statement

This chapter considers malicious actors have the capability to engage in passive reconnaissance or active probing to gather system information, including network topology and vulnerabilities of devices. An attacker has the ability to carry out their activities through compromised switches. Passive reconnaissance can help attackers acquire network traffic that flows through the switches and examine the network patterns to access details about devices, like IP addresses and weaknesses. The benefit of passive reconnaissance is its ability to gather information with minimal chances of being detected. Nevertheless, the downsides of passive reconnaissance involve its limited capacity to gather only partial information. Additionally, the passive nature implies a reliance on waiting to gather information, which may not always be valuable to the attackers.

Active probing enables the attacker to directly engage with specific targets in order to obtain particular information, such as requesting a server to disclose its open ports. This type of activity is effective and cost-efficient. Nonetheless, there is a high likelihood of being detected by the defender.

Having access to details about the network's users, devices, and services enables the creation of an attack plan that reduces the chances of being detected. For instance, by using information collected discreetly, an attacker could identify that their current position in the network cannot reach a critical server without setting off an alert. Consequently, they may decide to shift their position in the network to a user or device that has the capability to reach the server. Therefore, it is essential to protect against network reconnaissance.

## 4.4 System Model

### 4.4.1 Network Model

We consider a software-defined network (SDN)-based enterprise network consisting of servers, routers, and connected clients with a centralized entity. The SDN environment separates the network control and data planes (for example, packet forwarding) for greater flexibility, robust security/performance, and programmability [Scott-Hayward et al., 2013].

Considering each node's network position and worth (i.e., importance), we group all involved nodes within three subsets, including *asset layer*, *internal layer*, and *core layer*, as described in Fig. 4.1. Each layer is detailed as:

- *Asset Layer* (*AL*): This layer includes clients, such as Internet-of-Things (IoT) devices and laptops.
- *Internal Layer* (*IL*): This layer contains routers, switches, or other nodes between the asset layer and the core layers.
- *Core Layer* (*CL*): This layer includes high value nodes, such as database or other servers, which involve sensitive data.

Let $\mathcal{N} = N_{AL} \bigcup N_{IL} \bigcup N_{CL}$ denote the set of all nodes. We consider nodes in *IL* and *CL*. A node belonging to *IL* or *CL* is characterized by a *node worth*, denoted by $v(i)$, indicating the importance of the information node $i$ has, which is susceptible to attacks. Depending on which layer node $i$ belongs to, node $i$'s importance, $v(i)$, is determined differently.

*IL* nodes ($N_{IL}$) play a role in network flow transfer. Node $i$ is assigned with $W(i)$, representing node $i$'s workload (e.g., the number of network flows) where higher $W(i)$ is more valuable. *CL* nodes ($N_{CL}$) are the most valuable assets in a given network. Their value is defined based on their type, $T(i)$, such as database or web server, where $T(i) = \{\text{type}_1, \text{type}_2, \text{type}_3, \ldots\}, \forall i \in N_{CL}$. Each node $i$'s value, $v(i)$,

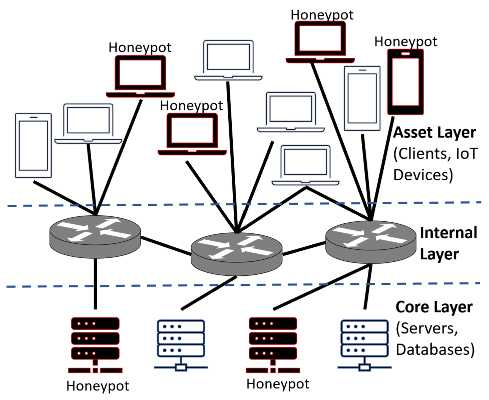


Figure 4.1: Network Model: 3-layer network.

is represented by these two values, $W(i)$ and $T(i)$, by:

$$v(i) = \begin{cases} W(i), & \text{if } i \in N_{IL} \\ T(i), & \text{if } i \in N_{CL} \end{cases} . \tag{4.1}$$

### 4.4.2 Defender Model

The defender uses various honeypots to collect the attackers' intelligence and protect tangible assets. Honeypots mimic fake vulnerable services to lure attackers into exploiting them. Virtual machines perform such a role with simple implementation; therefore, they are usually referred to as low-interaction honeypots, LHs. Moreover, adding LH nodes helps the defender hide the actual information (e.g., network topology or traffic data) that the attacker can collect using passive monitoring. The attackers can easily detect LHs. Therefore, for a node to look like a *CL* node (e.g., database servers), the defender needs to deploy HHs specifically within the core layer.

The defender is aware of the network topology and the values of the connected nodes. The defender aims to balance the different levels of honeypots (e.g., when to use LHs or HHs). LHs suffice to deceive passive attackers. However, they can be easily detected by active probing attackers. To prevent this, the defender can deploy HHs for interacting and responding to the attacker's probes and requests. Relying solely on HHs is impractical and incurs high costs. Moreover, deploying HHs is unnecessary if the defender thwarts passive reconnaissance. Therefore, the defender should be able to strategically select an optimal defense depending on a given network situation and the attacker's strategies.

**Deploy LHs** Low-interaction Honeypots (LH) [Seifert et al., 2007] are virtual machines that reside on a host. More specifically, LHs do not represent a fully-featured operating system and usually cannot be thoroughly exploited. As a result, an LH is not well suited for capturing active probings or zero-day exploits and can be easily detected by experienced attackers.

To protect against passive monitoring, the defender decides on a deception budget in terms of the number of honeypots to be allocated. Let $\mathcal{A}_d^\ell = [Low, Medium, High]$ be the strategy space for the defender, which represents three numbers of honeypots deployed in the asset layer and connected to *IL*. When the defender uses more LHs against passive reconnaissance and leads to a more secure internal and asset layer. The defender incurs a cost of $c_d$ associated with the implemented deception budget. However, to protect the core layer, the defender needs defense strategies that include HHs.

**Deploy HHs** High-interaction Honeypots (HH) [Wagener et al., 2009] represent real hosts attached to the network. Hence, we allow the defender to mimic more sophisticated systems that can interact with attackers and mislead them with false responses and fake information. Such a honeypot can monitor attackers and record their activities on the machine.

The defender deploys HHs to protect a particular device $i$. The strategy space containing the core layer nodes to be protected is denoted by $\mathcal{A}_d^h = N_{CL}$. An action $a_d \in \mathcal{A}_d^h$ defines a subset of the core layer nodes to be protected via HHs is defined as, $a_d = \{N_i | N_i \in N_{CL}\}$. HHs can successfully deceive attackers by performing active probing. However, the associated defender cost, denoted by $C_d$, is much higher, i.e., $C_d > c_d$. Therefore, the defender needs to balance between LHs and HH to be deployed to deceive the attacker. The defender action space is $\mathcal{A}_d = \mathcal{A}_d^\ell \times \mathcal{A}_d^h$.

### 4.4.3 Attacker Model

We consider an attacker in the reconnaissance stage, where the attacker targets an SDN-based enterprise network and aims to obtain the information, including the network topology and node worth, via passive monitoring and active probing.

**Passive Monitoring** The attacker performs passive monitoring to collect information about the network, such as hosts' IP addresses, connectivity, and hosts' worth. Passive monitoring snoops data exchanged in a network without altering it. Such attack includes traffic monitoring to identify communication parties and functionalities, eavesdropping, and traffic analysis [Goyal et al., 2010]. The attacks can help the attacker gain packet header and unencrypted information and learn the network topology and the connection between different hosts.

Assume an attacker must select one node in $IL$ (e.g., routers or switches) to perform passive monitoring and obtain information about the hosts only if their network traffic passes through the selected node. As a result, a higher workload of the selected $IL$ node means the attacker can obtain more information through the corresponding node. Recall $W(n)$ in Eq. 4.1 as the node worth of the nodes in $N_{IL}$. Because the attacker is unaware of the entire network topology at the beginning of the game, the attacker initially selects a random node as the target. After the attacker performs passive monitoring with a node, the attacker obtains the information corresponding to the node worth to the attacked node. We use $\mathcal{A}_a^p = N_{IL}$ to denote the action space when the attacker performs passive monitoring via node $N_i$ and incurs cost $c_a$.

**Active Probing** The attacker performs active probing by sending packets to a particular host to collect information regarding a specific host, such as finding open ports and OS type. Recall that the attacker's objective is to gather information from the core layer (e.g., database servers). Therefore, the active probing targets are the nodes in $CL$ denoted by $\mathcal{A}_a^a = N_{CL}$, which is the attacker's action space under active probing with associated cost $C_a$ where $C_a > c_a$.

The attacker action space is $\mathcal{A}_a = \mathcal{A}_a^p \times \mathcal{A}_a^a$. A pure action $a_a \in \mathcal{A}_a$ is the set of nodes to be probed actively and passively. The attacker balances the two reconnaissance levels to avoid excessive attack costs and gain necessary network information.

In practice, an attacker may discover honeypots. Let $\gamma$ and $\Gamma$ denote the attacker's probability of discovering LHs and HHs, respectively. Table 4.1 summarizes game parameters. The attacker's reward depends on the deception strategy implemented by the defender, as explained next in Section 4.4.4.

Table 4.1: Key Design Parameters, Meanings, and Their Default Values

| Symbol | Meaning | Default |
|---|---|---|
| $N_{AL}$, $N_{IL}$, $N_{CL}$ | The nodes in the asset layer, internal layer, and core layer, respectively | None |
| $W(i)$ | The node worth of the node in *IL* | [1,5] |
| $T(i)$ | The node worth of the node in *CL* | [6,10] |
| $r$ | Information leaked by passive monitoring | None |
| $R$ | Information leaked by active probing | None |
| $c_a$ | Cost of passive monitoring | 3 |
| $C_a$ | Cost of active probing | 10 |
| $c_d$ | Cost of low-interaction honeypot | [2,4,6] |
| $C_d$ | Cost of high-interaction honeypot | 10 |
| $v(i)$ | Value of an arbitrary node $i$ | Eq. 4.1 |
| $\gamma$ | Low-interaction honeypot detectability | 0.5 |
| $\Gamma$ | High-interaction honeypot detectability | 0.2 |

### 4.4.4 Utility Functions

Let $u_d$ and $u_a$ denote the utility functions for the defender and attacker, respectively. Consider a zero-sum game (i.e., $u_d + u_a = 0$) where the players action profile is $(a_d, a_a) \in \mathcal{A}$.

When the attacker performs passive monitoring with a node in the IL, $a_a \in \mathcal{A}^p$, and the defender deploys LHs, $a_d \in \mathcal{A}_d^{\ell}$. **The defender's utility when taking LHs** is expressed by:

$$u_d(a_d, a_a) = \Big(\sum_{i \in a_a} -[r \cdot \hat{v}(i)]\Big) - c_d \cdot a_d + c_a, \tag{4.2}$$

where $r$ is the information leaked via passive monitoring. Recall that $v(i) = W(i)$ is the value the node in *IL*. Let $W^{a_d}(i)$ denote the number of fake traffic generated by LHs and passing through node $i$ and $W(i)$ is the number of all traffic passing through node $i$. We assign $\hat{v}(i) = v(i) - W^{a_d}(i)$ as the value of node $i$ under deception due to fake traffic. In other words, the node's value is decreased due to deceptive traffic that belongs to honeypots.

Similarly, if the attacker performs active probing, (i.e., $a_a \in \mathcal{A}_a^a$), the defender needs to deploy HHs to mimic a certain type of node in *CL* (i.e., $a_d \in \mathcal{A}_d^h$). **The defender utility when taking HHs** is:

$$u_d(a_d, a_a) = \Big(\sum_{i \in a_a} R \cdot v(i) \mathbb{1}^{\Gamma}_{\{i \in a_d\}}\Big) - C_d \cdot |a_d| + C_a, \tag{4.3}$$

where $R$ denotes the leaked information due to active probing and $v(i) = T(i)$ is the value of a node $i \in a_a$ targeted via active probing. $\mathbb{1}^{\Gamma}_{\{\cdot\}}$ is a special indicator function that equals $\Gamma$, if $i \in a_d$, where $\Gamma$ is the probability of successful deception, and returns $-1$, otherwise. That is, if the attacked node is a honeypot, the defender receives a reward with the probability that the attacker is deceived successfully; the attacker gains a reward $R$, otherwise.

In some cases, the defender may implement a low level of deception against active probes, or high-level deception against passive monitoring. Thus, the utility will be a combination of Eqs. 4.2 and 4.3 as follows. In the first scenario, we have $a_d \in \mathcal{A}_d^{\ell}$ and $a_a \in \mathcal{A}_a^{a}$. As such, the defender utility is similar to Eq. 4.3, replacing $\Gamma$ by $\gamma$ as the probability of successful deception and replacing $C_d$ by $c_d$ as the cost of deception as given by:

$$u_d(a_d, a_a) = \Big( \sum_{i \in a_a} R \cdot v(i) \mathbb{1}^{\gamma}_{\{i \in a_d\}} \Big) - c_d \cdot a_d + C_a \tag{4.4}$$

In the second scenario, we have $a_d \in \mathcal{A}_d^{h}$ and $a_a \in \mathcal{A}_a^{p}$. The defender utility is:

$$u_d(a_d, a_a) = \Big( \sum_{i \in a_a} - [r \cdot \hat{v}(i)] \Big) - C_d \cdot |a_d| + c_a. \tag{4.5}$$

Both cases represent improper deception scenarios. However, the defender may fall into such scenarios due to a lack of information, which motivates our hypergame formulation as presented in the next section.

## 4.5 Hypergame Formulation

Considering the attacker and defender models above and the uncertainty associated with each player, we present our hypergame model. The game is played repeatedly between two players, each facing an interesting tradeoff. The attacker decides whether to use active or passive probes. The defender balances the use of LHs and HHs to reduce the cost and provide effective deception. The cost of running an HH is greater than that of an LH. If the defender is certain about the type of reconnaissance the attacker uses, it can better optimize its deception strategies.

### 4.5.1 Subgames

We develop a hypergame formulation that captures the possible subgames played by the attacker. We refer to the defender as the row player. We consider three subgames (i.e., subgame 1, 2, and 3) where each subgame specifies the set of strategies to be played by both players defined below:

1. *Subgame 1*: The attacker solely relies on passive monitoring to perform reconnaissance. The defender conducts deception via LHs or HHs. The action space of this subgame is $\mathcal{A}_d^{l} \times \mathcal{A}_a^{p}$. The defender decides on the deception budget to invest in deception. The attacker selects a switch/router to monitor for a specific duration to gather information about the nodes' connectivity, and traffic flows passing through the targeted switch. Deception via *LH* is considered dominant for the defender in this subgame as *HH* incurs unnecessary costs.

2. *Subgame 2*: The attacker uses active probing to gather information about a node in the *CL*. The defender uses an HH to protect a certain node in the *CL*. The action space of this subgame is $\mathcal{A}_d^{h} \times \mathcal{A}_a^{a}$.

3. *Subgame 3*: This is a full game. Therefore, the action space of this subgame is $\mathcal{A}_d \times \mathcal{A}_a$.

### 4.5.2 Players' Beliefs

The belief vector is $\mathbf{P} = [P_1, P_2, P_3]$, where $P_3 = 1 - P_1 - P_2$. $P_k$ is the true probability that subgame $k$ will take place. These probabilities will be given based on the attacker's preference in performing different reconnaissance approaches (i.e., passive, active, or both). In practice, we assume that these probabilities are initially unknown to the defender. The defender improves his expected utility by learning the attacker's subgame preference $P_k$ as in [House and Cybenko, 2010]. In this work, we assume that the defender knows the attacker's subgame preferences $P_k$s with uncertainty $g_d$ as detailed next.

### 4.5.3 Players' Uncertainty

The defender is uncertain about which subgame the attacker will play and thus does not know true $P_k$'s. However, as the game is repeated over time, the defender can learn the actual belief probability $P_k$ for each subgame. One way is to assume that the defender observes the actual probability distribution of $P_k$'s with probability $1 - g_d$ where $g_d$ is the defender's perceived uncertainty toward what subgame the attacker will play. Hence, the defender is assumed to know the actual belief vector with $1 - g_d$. $g_d$ is represented by a decay function in terms of the number of play rounds $j$ as follows:

$$g_d(j) = \exp(-\frac{j}{\mu}), \tag{4.6}$$

where $\mu$ is a decay rate.

Let $g_a$ denote the attacker perceived uncertainty about the cyberdeception implemented by the defender. $g_a$ exponentially decays in terms of $\Gamma$ (i.e., HH detectability by the attacker), and $\gamma$ (i.e., LH detectability by the attacker) by:

$$g_a(\gamma, \Gamma) = \exp(-\frac{\gamma + \Gamma}{2}). \tag{4.7}$$

### 4.5.4 Players' Mixed Strategies

If the attacker is playing *subgame 1* (i.e., performing passive monitoring), it is more efficient for the defender to implement deception using LHs. In fact, implementing HHs will come at an excessive cost for the defender (i.e., dominated strategies by $\mathcal{A}_d^\ell$ strategies). However, if the attacker performs active probing (i.e., *subgame 2*), the defender needs to use HHs and thus $\mathcal{A}_d^h$ will be the dominant strategies. Finally, if the attacker plays a combined reconnaissance (i.e., *subgame 3*), which is a full game), the defender needs to use both levels of deception. We consider a rational attacker that plays Nash equilibrium strategies within each subgame.

For any subgame $k \in \{1, 2, 3\}$, the defender (i.e., row player) mixed strategies, $DEF_k = [d_{k1}, \dots, d_{km}]$, where $m = |\mathcal{A}_d|$ for the full game, such that $\sum_{i=1}^{m} d_{ki} = 1$. Similarly, the attacker (i.e., column player) mixed strategies at the $k^{th}$ subgame is believed by the defender to be $Att_k$ such that, $Att_k = [\mathrm{a}_{k1}, \dots, \mathrm{a}_{kn}]$, where $n = |\mathcal{A}_a|$ for the full game, such that $\sum_{j=1}^{n} \mathrm{a}_{kj} = 1$.

According to the belief vector $\mathbf{P}$, the defender redefines its beliefs regarding the mixed strategies played by the attacker at different subgames. Specifically, let $\bar{\mathbf{a}} = [\bar{\mathrm{a}}_1, \dots, \bar{\mathrm{a}}_n]$, where $\bar{\mathrm{a}}_j = \sum_{k=0}^{4} P_k \mathrm{a}_{kj}$ for $\forall j = 1, \dots, n$.

### 4.5.5 Hypergame Expected Utility

Now we can calculate the hypergame expected utility (HEU) for the defender. Recall that the defender is uncertain about its beliefs regarding the subgame. Hence, the defender's HEU (DHEU) is a weighted combination of the EUs obtained under its belief and the EU under uncertainty, $g_d$. When the defender takes $a_d$, DHEU is obtained by:

$$DHEU(a_d, g_d) = (1 - g_d) \cdot EU\left(a_d; \bar{\mathbf{a}}\right) + g_D \cdot EU(a_d; \mathrm{a}_w), \tag{4.8}$$

where $EU\left(a_d; \bar{\mathbf{a}}\right) = \sum_{a_a \in \mathcal{A}_a} \bar{\mathrm{a}}(a_a) u_d(a_d, a_a)$. The second term represents the worst-case expected utility received by the defender due to uncertainty when playing a strategy $a_d$ against a damaging attack strategy $w$, such that $EU(a_d; \mathrm{a}_w) = n \cdot \mathrm{a}(a_w) \cdot u(a_d, a_w);\ a_w \in \mathcal{A}_a$.

The defender selects the defense strategy that maximizes his hypergame expected utility, $DHEU$. The attacker's mixed strategies, $\mathrm{a}_{kj}$'s, are considered to represent Nash equilibrium mixed strategies within each subgame.

Similarly, the attacker's HEU (AHEU) for any strategy $a_a$ is given by:

$$AHEU(a_a, g_a) = (1 - g_a) \cdot EU\left(a_a; \bar{\mathbf{d}}\right) + g_a \cdot EU(a_a; \mathrm{a}_w). \tag{4.9}$$

## 4.6 Evaluation by Simulation and Results

**Simulation Settings.** Recall that each node belongs to one of the three layers. In this simulation, we consider 100 nodes belonging to the asset layer, 70 nodes in the internal layer, and the core layer containing 10 nodes. Nodes in the asset layer $AL$ are assigned a value between $[1, 5]$ and between $[5, 10]$ for the node in the core layer $CL$. The IL nodes' values are calculated via Eq. 4.1. When the defender deploys low-interaction honeypots, each honeypot obtains an importance value of $[1, 5]$. If the defender deploys a high-interaction honeypot strategy to protect a particular node in CL, we create a new node as the honeypot and assign it the same importance value as the protected node. For attackers, we assign LH and HH detectabilities with $\gamma = 0.5$ and $\Gamma = 0.2$, respectively. Table 4.1 summarizes the notations of key design parameters, the corresponding meaning, and default values.

We use the following performance metrics:

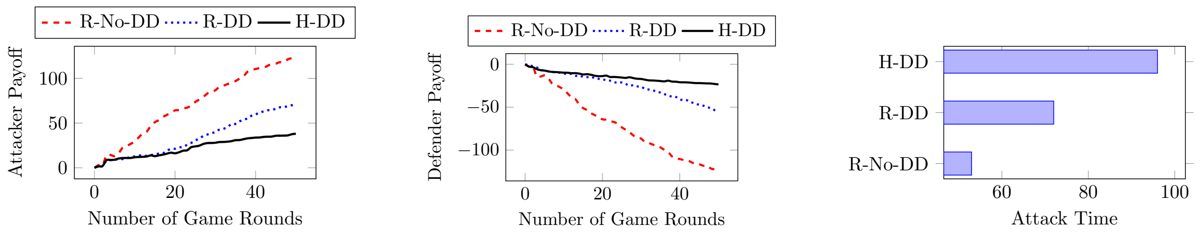
- **Accumulated Payoff:** This is the accumulated values of an attacker's and defender's payoffs with respect to the number of game rounds, respectively.

- **Total Attack Time**: The attacker aims to collect all the values of the nodes in the $CL$. We use the number of game rounds for the attacker to complete its objective successfully to estimate this metric.

We compare the following policies for the defender and attacker to choose their best strategies:

- **Random with No DD (R-No-DD)**: Given no knowledge of the network, an attacker randomly selects its action when the defender does not use any defensive deception (DD).

- **Random with DD (R-DD)**: The defender and attacker randomly select their strategies when the defender uses DD.



(a) Attacker's payoffs under various schemes with respect to the number of game rounds.

(b) Defender's payoffs under various schemes with respect to the number of game rounds.

(c) Total attack time with different schemes.

Figure 4.2: Comparative performance analysis of defensive deception vs. non-defensive deception under R-No-DD, R-DD, and H-DD.

- **Hypergame with DD (H-DD)**: The attacker and defender play a hypergame when the defender uses DD.

Fig. 4.2 (a) shows the attacker's accumulated payoffs when the attacker and defender play under the three policies (i.e., R-No-DD, R-DD, H-DD). Similarly, Fig. 4.2 (b) shows the defender's accumulated payoffs. Under R-No-DD (i.e., red dashed line), the attacker's payoff is significantly higher than under R-DD and H-DD (i.e., blue dotted and black solid lines). When the defender uses DD, the attacker's payoff decreases. Even if the defender randomly selects its strategies, the honeypots provide false information (e.g., deceptive network flow and fake nodes) and disturb the attacker's reconnaissance, as described in Eqs. 4.2 and 4.3). Moreover, playing H-DD decreases the attacker's payoff and increases the defender's payoff compared to playing R-DD. Recall that the uncertainty creates a discrepancy between the players' beliefs about the game. The defender uncertainty ($g_d(j)$) decreases over game rounds, which implies that the defender knows better about its opponents and chooses more suitable strategies against the attacker. As a result, the defender obtains the highest payoff under H-DD.

Fig. 4.2 (c) shows the number of game rounds (such as the total attack time) exhausted by the attacker to obtain all the values of the core layer nodes successfully. Without DD, the attacker needs 53 game rounds to harvest the network information and the core layer nodes' worth. LHs and HHs provide false information to delay the attacker's passive monitoring and active probing (i.e., R-DD and H-DD). In addition, hypergame enables the defender to identify the best response to the attacker's reconnaissance actions (i.e., H-DD) where real-world uncertainties are better reflected on H-DD.

## 4.7 Conclusion

This research proposed a hypergame-based hybrid honeypot system to defend against malicious reconnaissance. We proved that defensive deception could significantly delay the attacker's processes, leading to attack failures. We also observed that the hypergame considering uncertainty in practice enables the defender to select optimal strategies to mislead the attacker's reconnaissance by leveraging the inherent uncertainty that can mislead the attacker's perception.

CHAPTER

5

# GANFLOW: DECOY NETWORK TRAFFIC GENERATION

This chapter proposes a Generative Adversarial Network (GAN)–based decoy network traffic generator named GanFlow. We demonstrate that a honey controller learns real hosts' behavior patterns and guides honeypots to produce realistic TCP-based network traffic. Considering passive monitoring as the threat model, we show that our GamFlow generator can assist a honeypot in imitating a particular host in terms of the frequency of packets generated and the time duration of TCP connections.

## 5.1 Introduction

Adversarial network reconnaissance helps steal confidential and lucrative military, financial, and business-related information transferred within the network by current techniques that enable analyzing network traffic and determining the weakest points of a computer network. Attackers can conduct active or passive reconnaissance [Bartlett et al., 2007] to gather important information from the network, which helps them choose a target to attack. Specifically, an attacker can apply banner grabbing to reveal compromising information about the services running on a host or use packet sniffing to capture network packets for later analysis. Any data sent in plain text, such as user names, passwords, IP addresses, and other sensitive data, is vulnerable to eavesdropping. A skilled attacker can engage with a network environment in multiple phases and stay for an extended period to conduct network scanning. And attacker can compromise critical resources (e.g., workstations and servers) and control several essential components of a network.

Consider a Software Defined Networking (SDN) environment. A compromised SDN switch is especially dangerous since attackers can use it to perform passive reconnaissance [Anjum et al., 2021]. Network traffic analysis provides advantages to the attackers in passive reconnaissance since they can intercept and analyze network packets without engaging with the environment, potentially lowering the risk of being detected. As a result, defenders may receive little to no notification that a malicious reconnaissance is underway.

As network infrastructure grows, attackers take advantage of learning from a dynamic environment. Therefore, typical network security management systems such as intrusion detection systems (IDS) and firewalls are insufficient for preventing malicious network reconnaissance. Cyberdeception techniques using decoy objects can effectively reduce the quality of the hostile reconnaissance by providing false information to attackers [Zhu et al., 2021a]. For example, network defenders can install fake hosts (honeypots) to mislead attackers and make them hesitant to continue additional attacks. They generate fake traffic to support honeypots or conceal natural traffic properties on their networks. Therefore, assisting honeypots in imitating actual hosts is an important challenge for honeypot research. Specifically, sending fake traffic that mimics real traffic can invalidate attackers' gathering of information in the passive reconnaissance phase and confuse them in identifying network vulnerabilities. Almotairi et al. [2009] present a technique for detecting attacks via low-interaction honeypot traffic. They apply principal component analysis to analyze network traffic features. Nazario [2009] proposes a design and implementation of PhoneyC, which is a client program that imitates an HTTP client. We demonstrate *GanFlow*, a novel decoy traffic generator for TCP-based fake flows to mimic legitimate client and server behaviors.

Recently, Generative Adversarial Networks (GANs) [Goodfellow, 2016] have shown great promise in generating deceptive objects, focusing primarily on images and video applications. The model relies on a pair of neural networks: a generator and a discriminator. The generator is trained to map from a latent space to a particular data distribution, and the discriminator distinguishes candidates produced by the generator from the actual data distribution. The objective of the generator is to increase the error rate of the discriminator where the neural networks play a zero-sum game. We adopt a GAN model to mimic natural networks and generate fake flows to mislead attackers.

### Structure of This Chapter

Section 5.3 describes our problem in terms of details about passive monitoring, the considered network configuration, and the assumptions. Section 5.4 demonstrates the design of our deception architecture and the GAN model. Section 5.5 shows the implementation, simulation, and results. Section 5.6 is the conclusion and future work.

## 5.2 Related Work

Machine learning (ML) has been extensively employed in deception studies to help the defender find the most effective approach to enhance the efficiency of deception techniques and reduce the cost to the defender. Li et al. [2022] suggest a defensive deception framework to protect cloud tenant networks from reconnaissance attacks. They create a utility function to simulate the OS common vulnerability-associated threat scenario for the cloud and then customize a DRL agent to determine the optimal deception strategy based on the utility function. Zhu et al. [2022] design a deceptive network flow via the generative adversarial network (GAN) to disrupt the attacker's reconnaissance. Charpentier et al. [2022] explore an attacker-defender model that takes into account the asymmetry of the players' understanding and the distinction between MTD and deception tactics. They use Deep Q-Learning to optimize the selection of defense strategies against an attacker. Ring et al. [2019] and Cheng [2019] both employ Generative Adversarial Networks (GANs) to generate realistic network traffic that can be transmitted over the Internet and elicit expected responses from the network. However, neither of these works takes *time* into account as an attribute. While some works have considered time as an attribute when generating new data using GANs, they express timestamps using numerical values such as the number of seconds past Epoch (January 1, 1970) [Patki et al., 2016], or generate irregularly sampled time series by conditioning the generator and the discriminator with the timestamps [Ramponi et al., 2018]. In contrast, we consider network data attributes similar to utterances in language processing and use deep learning approaches to process continuous and categorical attributes.

## 5.3 Problem Statement

Targeted attacks usually begin with the attacker gaining a foothold in a network and conducting reconnaissance for selecting potential targets to intrude. A malicious actor plans a targeted attack by collecting information and initiating an Advanced Persistence Threat (APT) to intrude on a system.

This work concentrates on an attacker performing passive reconnaissance (e.g., packet sniffer) through compromised switches or routers to collect network traffic for selecting vulnerable and valuable targets. Specifically, an attacker gathers network traffic by capturing packets that pass through a compromised switch and storing the data for later analysis. By collecting information on client-server identity to plan attacks, the attacker aims to detect valuable and vulnerable devices.

#### Network Configurations

We consider an SDN-based network consisting of a centralized controller, servers, workstations, Openflow switches, and routers. In addition, the network includes fake hosts (e.g., honeypots) to enhance cybersecurity. The main objective of honeypots is to attract attackers' attention and protect actual hosts. Each network node may have pre-existing vulnerabilities in the OS and software, and these vulnerabilities are generally exploitable. The defender's inventory may indicate the existence of a vulnerable server, but due to production requirements, some devices are not yet patched.

In our model, a *vulnerability value* represents how difficult it is to compromise a host when the host contains a particular type of vulnerability. Every host in the network obtains a vulnerability value based on its actual situation, such as the OS type and application versions. For example, suppose a host deploys an old version of Windows. In that case, it will get a high vulnerability value, which means the attacker can exploit it with less effort or resources. Additionally, the same vulnerability can have different costs for the defender. We assign an *importance value* to each host from the perspective of the defender and based on the type of devices (e.g., servers, laptops, and IoT devices) and the roles of the device owners (e.g., CEO or officer). For example, a server or database in the SDN network should be more valuable than a laptop that has no confidential information.

#### Assumption

We assume that the attacker attempts to identify targets based on communications between clients and servers. Assume the attacker has control of at least one compromised switch and can capture packets and execute necessary analysis only during the reconnaissance phase. We assume the attacker has the necessary processing power to analyze its target information, including vulnerability and importance values, based on the observed network flows.

The trusted computing base (TCB) includes the system defender (SDN controller or a separate trusted server) and the southbound network between the SDN controller and switches. We do not assume SDN switches are trustworthy, but we assume that not all the switches are compromised. We focus on passive traffic monitoring without discussing Openflow switch compromise methods and other malicious actions in the SDN environment. In addition, we only consider TLS-protected TCP communications. However, research shows that encrypted web traffic can leak information through packet length, timing, web flow size, and response delay [Schuster et al., 2017].

## 5.4 Deception Architecture

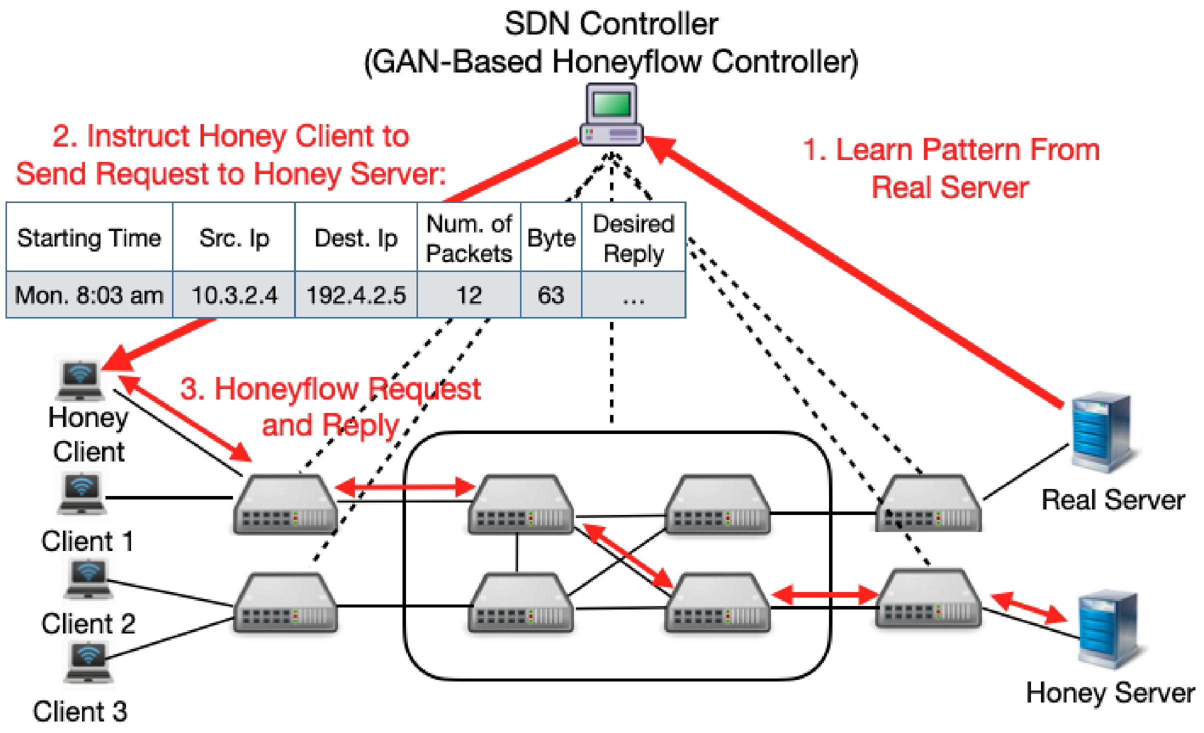
Network vulnerabilities constantly charge, leaving new weaknesses in the network for an attacker to exploit the system. Monitoring and patching vulnerabilities of all the nodes for the entire network require intense resources and might not be cost-effective. The defender's goal is to strategically deploy honeypots and corresponding decoy network traffic to reveal fake vulnerabilities and importance to confuse the attacker and lure it towards a fake host. The challenge is how to create a realistic enough decoy flow that can mislead attackers. Here, we propose how to create GanFlows, that provide false information and mislead the attacker.

We consider all fake clients and servers connected with switches within the network. Leveraging the advantage of the SDN environment, the defender can collect and monitor all the connected clients' and servers' network flow. Monitoring and capturing specific flows passing through the network, especially vulnerable servers, is vital before implementing GanFlow. The defender's observation might include the following:

- Regular Traffic Flow: Information that the SDN controller and network administrator have, such as the flow rules and the numbers of network flows between nodes.

- GanFlow: Information about the GanFlow including the routing path, count, and contents of the GanFlow.

- Attack Detection: We assume if an attacker trusts the misinformation in a GanFlow and deploys attacks based on the misinformation, the defender can detect the attack.


SDN Controller
(GAN-Based Honeyflow Controller)
2. Instruct Honey Client to Send Request to Honey Server:
1. Learn Pattern From Real Server
3. Honeyflow Request and Reply
Honey Client
Client 1
Client 2
Client 3
Real Server
Honey Server

| Starting Time | Src. Ip | Dest. Ip | Num. of Packets | Byte | Desired Reply |
|---|---|---|---|---|---|
| Mon. 8:03 am | 10.3.2.4 | 192.4.2.5 | 12 | 63 | ... |

Figure 5.1: Overview of GanFlow Generation. (1) The GanFlow controller learns access patterns of the real server. (2) The GanFlow controller guides the honey client via instructions that include information such as starting time, source IP, destination IP, number of packets in the GanFlow, bytes in each packet, and service type; 3) The honey client sends requests to the corresponding honey server by following the instructions.

Based on these observations, the defender determines the optimal deception strategies, including how many fake hosts to install and what GanFlow to develop. Generally, network utilization is not near-maximum capacity during regular operation. However, excessive GanFlows may cause congestion and cause performance degradation.

Our deception architecture has four parts: (1) honey servers, (2) honey clients, (3) a honey controller, and (4) GanFlows. The honey controller collects legitimate hosts' network features to train a GAN generator for instructing honey servers and honey clients to imitate actual hosts and generate the GanFlows between them.

### 5.4.1 Honey Controller

Within an SDN environment, the SDN controller can obtain network flow information, such as source/destination hosts, the number of transferred packets, and the bytes of packets. Leveraging this feature, we propose the honey controller, located in the SDN controller, to learn from the hosts and generate instructions that lead honeypots to create fake connections. For luring and misleading attackers, honeypots need to imitate actual hosts and release false information of two types.

**Network-level false information** is applied to guide a honeypot to mimic regular hosts' behaviors. Specifically, to mimic a regular host, we use six features to parameterize the creation of GanFlow connections: (1) date first seen, (2) duration, (4) source/destination IP addresses and port numbers, (5) number of packets, and (6) number of bytes in each packet. To generate realistic honey connections, a honey connection generator needs to learn the real server's communications pattern.

**Application-level false information** includes the details that assist attackers in searching for targets and preparing their exploitation. Specifically, an attacker has a belief about one device's importance and vulnerabilities based on such information. For example, HTTP/HTTPS connections may include the server's OS information. Other application-level information involves cookies and passwords for unencrypted protocols.

### 5.4.2 Honey Servers and Honey Clients

Servers and clients have distinct usage patterns, e.g., what times of day they receive more or fewer requests. Honey servers and honey clients need to exchange packets in accordance with such patterns.

We apply honeypots to imitate actual servers to receive TCP requests from honey clients. We identify each honey server with a tuple: $\langle \mathrm{IP}, \mathrm{MAC}, \mathrm{Role}, \mathrm{Vulnerability} \rangle$, where IP and MAC represent the honey server's addresses, Role indicates (e.g., web server and database server), and Vulnerability specifies what vulnerability information involves. Honey clients are honeypots that imitate regular clients.

### 5.4.3 GanFlow

A GanFlow represents communication between honey clients and honey servers. For simplicity, we focus on the TCP connections in this chapter.

To mislead passive monitoring attacks, a GanFlow must imitate a regular network flow and involve false information. Therefore, we propose a GAN model to learn from real hosts' behavior patterns and guide honeypots' accordingly. Specifically, we apply a GAN generator to learn the real traffic between a server and a client to instruct fake servers and fake clients. As TCP-based connections include request and reply, we propose *honey request (HQ)* and *honey reply (HR)* to capture this feature. The honey controller learns the hosts' network behavior and instructs the honeypots to imitate real servers and clients, as shown in Figure 5.1. Specifically, *Starting Time* indicates the detailed time for sending the request, *Number of Packets* represents how many packets the honey client will send in this connection, *Byte* denotes the number of bytes in each packet, and *Desired Reply* means how the corresponding honey server should reply the request.

**Architecture for GanFlow Generation**

Based on the GAN model, we identify three components: (1) GanFlow Generator, (2) Environment: all normal traffic is the real data, and (3) Passive Monitoring Attacker, following the concept of discriminator in GAN.

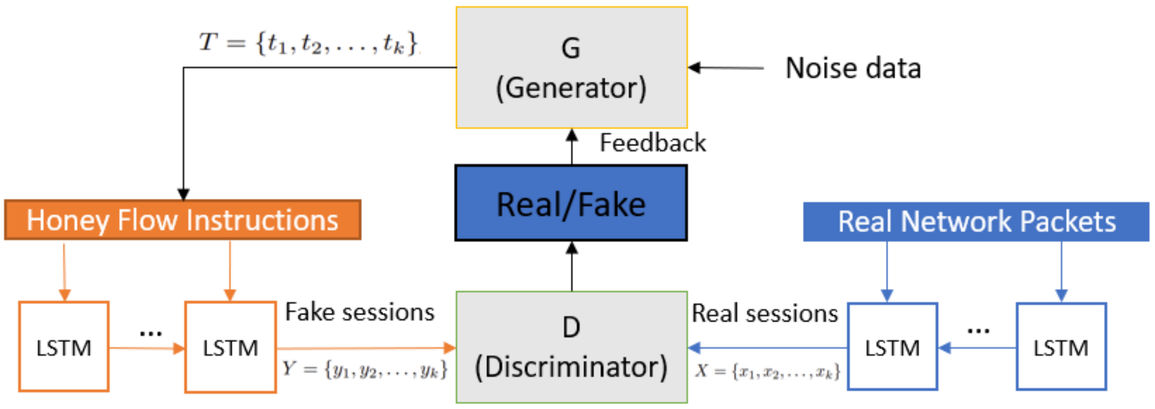


Figure 5.2: AE-LSTM model architecture. The input of the AutoEncoder are the packets along with the attributes defined in Section 5.4.1. We decode reconstructed packets from generator(G) to guide the honey connection. The AutoEncoder is fully connected with an LSTM network, the output of the encoder is the input of the discriminator (D). z represents the noise to train the generator.

The GAN model is trained to generate its vectors in the latent space; those vectors are decoded to produce a realistic network flow. A standard GAN can process only continuous input attributes. This is a major challenge since network flow data have categorical attributes (e.g., *IP addresses or port numbers*). Another challenge is how to interpret continuous output vectors. Therefore, we conceive of network flow attributes as analogous to **utterances** in natural language (NL). We adopt deep learning approaches for NL generation to overcome the above challenges. LaTextGAN [Donahue and Rumshisky, 2018] uses an AutoEncoder to learn a low-dimensional representation of sentences and generate sentences with the improved Wasserstein GAN (WGAN) [Gulrajani et al., 2017]. The model architecture is shown in Figure 5.2. We adopt the AutoEncoder (AE) and Long-Short Term Memory (LSTM) model for representing the attributes of the packets. Moreover, we use the AE–LSTM model to encode the categorical attributes into smooth representations. The generator network is then trained to generate its own flow-based representations in the learned latent space. Each network flow-based data vector produced by the generator is then passed through the decoder, which decodes to the nearest network attributes.

### 5.4.4 Feature Embedding

We adopt an AutoEncoder component that learns a dense high-dimensional (e.g., 200-dimensional) representation of the network flow data. The encoder network compresses information about each attribute into a vector. The decoder network reconstructs the input representation from the vector. LSTM networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. We aim to learn the pattern of realistic network traffic. Therefore, we adopt the LSTM network for both the encoder and the decoder. During the reconstruction, the decoder takes the latent representation and the previous hidden state as input and produces a probability distribution. We select the highest probability next state sequentially.

We now interpret the compressing and reconstructing processes of the encoder and decoder as shown in Figure 5.2. Let $X = \{x_1, x_2, \ldots, x_k\}$, where $x_i$ represents the attributes of packet-level records, be a real session, as in Figure 5.2. Let $T = \{t_1, t_2, \ldots, t_k\}$, where $T$ represents the output of the generator that needs to be reassembled as fake sessions as the input data of the discriminator. The decoder reconstructs the data according to the characteristic sequence $T$. Let the reconstructed sequence be $Y = \{y_1, y_2, \ldots, y_k\}$. The encoding and decoding processes follow these equations:

$$t_i = f(w_t x_i + b_t) \tag{5.1}$$

$$y_i = g(w_y t_i + b_y) \tag{5.2}$$

where $f\ (\cdot)$ and $g\ (\cdot)$ are ReLU functions, $w_t$, $w_y$ are weights, and $b_t$, $b_y$ are biases. We train the AutoEncoder by minimizing the cross-entropy loss in the reconstructing process.

### 5.4.5 Adversarial Training

The generator produces additional points in the latent space, which decode to valid packets. Typically, a discriminator network is trained to classify real and generated packets from their latent representations. The generator is the defender attempting to fool the discriminator by generating realistic packets. The discriminator distinguishes the attacker's passive reconnaissance processes.

We adopt a synthetic flow-based network traffic generator model based on Improved Wasserstein GAN (WGAN) [Gulrajani et al., 2017]. WGAN improved the original model by using gradient penalty as a soft constraint to enforce the Lipschitz constraint. It also applies the training objective to large ResNet architectures [He et al., 2016] to mitigate gradient instability associated with the randomly initialized fully connected layers. We train the WGAN with the output of the AutoEncoder $t$ in Equation 5.1 as the input in Equation 5.3. Here, $z$ represents the noise data with the same dimension as the finite vector generated by the AutoEncoder.

$$\max_{\theta} E_{z,p(z)} = [f_w(g(\theta(z)))] - E_{t,p(t)}[f_w(g(\theta(t)))] \tag{5.3}$$

Upon training the generator, we use it to guide GanFlow generation and build the node model with fake traffic.

## 5.5 Implementation

This section intends to provide a thorough overview of our methodological approach to utilizing GAN technology, covering the implementation process from data selection and curation to model training and evaluation.

### 5.5.1 Data Curation

Table 5.1: Overview of the attributes selected from the CIDDS-001 dataset.

| Attribute | Type | Example | Description |
|---|---|---|---|
| Date first seen | timestamp | 2017-03-15 | time flow first seen |
| Duration | continuous | 0.12 | Time differences of a flow's first and last packets |
| Transport protocol | categorical | TCP | Transport protocol |
| Source IP address | categorical | 192.168.100.5 | Source IP Address |
| Source port | categorical | 52128 | Source port |
| Destination IP address | categorical | 8.8.8.8 | Destination IP Address |
| Destination IP port | categorical | 80 | Destination port |
| Bytes | numeric | 2391 | Number of transmitted bytes |
| Packets | numeric | 12 | Number of transmitted packets |
| TCP flags | binary/categorical | .A..S. | Indication of a particular state of TCP connection |

The CIDDS-001 dataset [Ring et al., 2019] is commonly utilized in research focusing on network-based intrusion detection [Althubiti et al., 2018; Verma and Ranga, 2018]. To generate the CIDDS-001 dataset, the authors simulated a small business setting with various clients and standard servers such as an E-Mail server and Web servers (refer to the figure nearby). Python scripts were employed to mimic legitimate user actions such as web browsing, email communication, and file sharing. To replicate authentic user behavior, each client follows an individualized work schedule that includes regular work hours and lunch breaks. Furthermore, the authors considered that employees with distinct roles engage in different types of tasks. For example, a manager may participate in more meetings, resulting in lower network activity compared to a researcher who frequently browses the web. Consequently, specific user attributes are defined through a configuration file [Ring et al., 2017a,b].

In our study, we utilize the CIDDS-001 dataset, which contains attributes of flow-based packets. Specifically, we choose features such as the initial timestamp, duration, source IP, destination IP, bytes, and packets from this dataset. A depiction of the characteristics of the dataset that we utilized is presented in Table 5.1. By analyzing the total count of packets, bytes, and duration, we can model client behavior to support the creation of honey connections. Randomly selecting 10 clients and one server from the CIDDS-001 dataset, these clients belong to the developer subnet, while the server acts as the File Server within the same subnet. The clients establish connections to shared folders on the File Server by mapping them as network drives. For each client, we monitor all traffic exchanged with the File Server over a week and generate a corresponding set of honeyflows. Similarly, for the File Server, we log all packets it transmits in response to client requests as genuine data, mimicking the patterns in the created honeyflows. The table in this section details the volumes of packets sent from each client to the File Server (192.168.100.5) during the week, as depicted in Table 5.2.

The generator in the GAN acquires data from servers and clients, then proceeds to generate fake flows by analyzing network flow patterns. As depicted in Figure 5.3, similarities can be observed in the transmission patterns of packets from the server to clients in both the actual data (bottom) and the simulated data (top). In Figure 5.3 (a), the graph illustrates the average bytes of flows from clients to the server over active time. Similarly, Figures 5.3 (b) and (c) represent the average packets of flows from clients to the server and the average time intervals between the first and last packets sent. Notably, the server exhibits activity during daytime and remains inactive from 9:00 pm to 5:00 am. Our research

Table 5.2: Overview of the total number of flows each client sent in a week.

| Client IP | Day1 | Day2 | Day3 | Day4 | Day5 | Day6 |
|---|---|---|---|---|---|---|
| 192.168.220.6 | 1913 | 1122 | 1355 | 1310 | 1304 | 665 |
| 192.168.220.7 | 2492 | 2274 | 2524 | 1288 | 792 | 1524 |
| 192.168.220.8 | 1627 | 1321 | 1431 | 1122 | 1047 | 1045 |
| 192.168.220.9 | 1635 | 1299 | 1674 | 1162 | 1083 | 1070 |
| 192.168.220.10 | 1713 | 1303 | 1287 | 1288 | 1188 | 870 |
| 192.168.220.11 | 1566 | 1233 | 1374 | 1274 | 1324 | 841 |
| 192.168.220.12 | 1595 | 1643 | 1532 | 132 | 1011 | 772 |
| 192.168.220.13 | 1539 | 15236 | 1443 | 1187 | 1235 | 685 |

validates the successful generation of packet-level honeyflows by our model.

### 5.5.2 Comparison between Real and Simulated Data

*The Independent Samples t test* [Kim, 2015] is employed to assess if a notable distinction exists between two distinct samples. This statistical method relies on various assumptions, with a key one being that the populations from which the samples are taken follow a normal distribution, and another requirement being that the variances of the two samples are equivalent. Here is an elaborate description of the independent sample t test, along with the mathematical expressions utilized: Independent Samples t-statistic: The t-statistic for independent samples gauges the variation between the means of two samples. Its computation is done using the following formula:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{5.4}$$

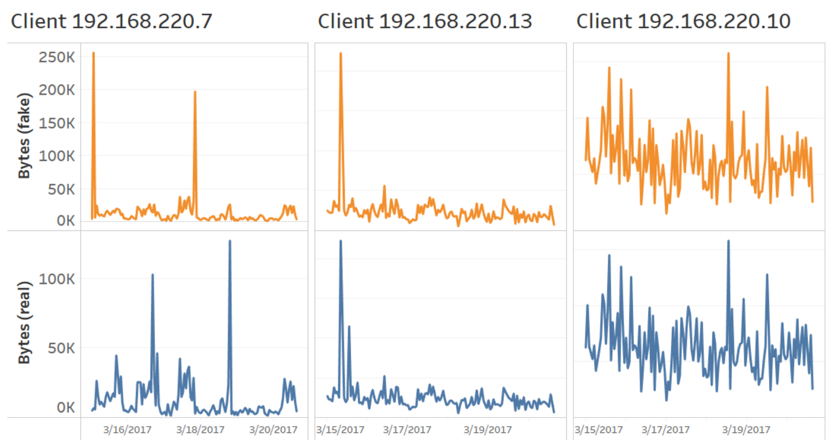
, where: $\bar{x}_1$ and $\bar{x}_2$ are the means of the two samples, $n_1$ and $n_2$ are the sizes of the two samples. , and $s_p$ is the pooled standard deviation, calculated as follows:

$$s_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}} \tag{5.5}$$

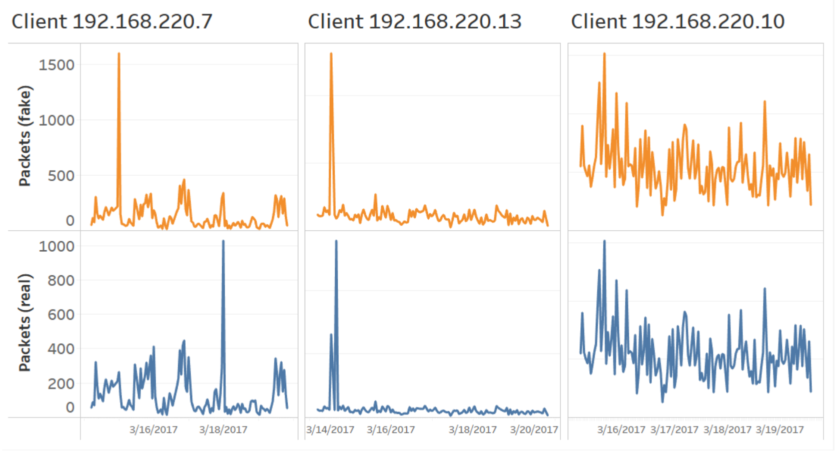
,where $s_1$ and $s_2$ are the standard deviations of the two samples.

We assess the effectiveness of our GAN by comparing the generated traffic with real client behavior. To gauge the similarity between the two datasets, we use the t-test as a metric. It is worth mentioning that none of the studies reviewed in Section 5.2 employed the t-test to evaluate GAN performance. Nevertheless, the experimental setup in our research differs from previous works. While traffic in enterprise networks does not typically adhere to a normal distribution, Lumley et al. (2002) have demonstrated the validity of the t-test for any distribution in large samples [Lumley et al., 2002]. Since our goal is to model client behavior, especially their activities during working hours, we categorize their actions hourly and compute the average bytes, packets, and packet durations transmitted to the file server.

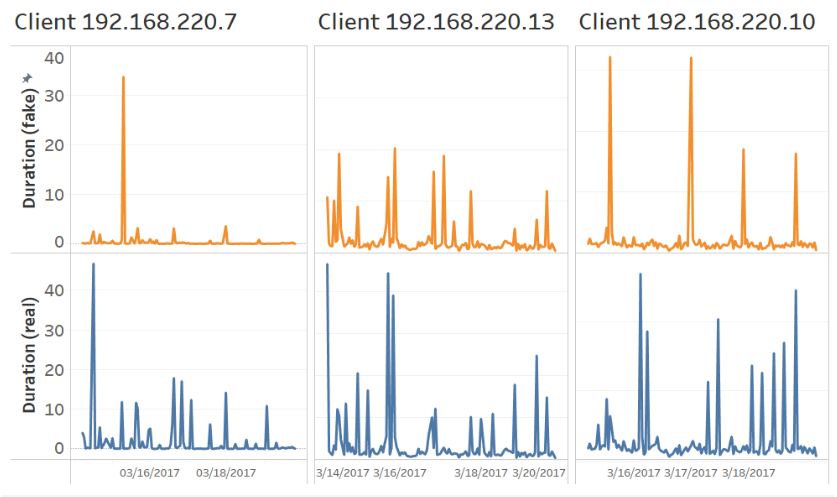
We conduct the t test on the original flows and the generated flows. To apply the t test, we remove the outliers of each attribute for each client in the same way of the plots in Figure 5.3. To do so, we manually review the real flows in the dataset and remove very rare and very large values, for example, 10 real flows have 109,827 packets for client 192.168.220.4. Then we measure the p-value between the



(a) The y-axes shows the average bytes of the flows from clients to the server.



(b) The y-axes shows the average packets of the flows from clients to the server.



(c) The y-axes of shows the average time differences (duration) between the flows' first and last packets sent.

Figure 5.3: The x-axes of the figures represent the sending times of the flows. The IP address of each client is on the top of each figure. We calculate the average of bytes, packets, and durations of the flows each client sent.

average number of attributes each client sent to the server per five hours and the corresponding generated flows. We take five hours as the unit here because we consider it is a favorable time slot to learn the clients' behaviors. We expect the p-value of the t test being high to mean that the null hypothesis, the true difference between means of real data and means of fake data are zeros, is not rejected. Table 5.3

Table 5.3: Results of the t test for clients receiving packets and server sending packets

| | p-value of clients | | | p-value of server |
|---|---|---|---|---|
| Client IP | Bytes | Packets | Duration | Frequency |
| 192.168.220.4 | 0.35 | 0.34 | 0.32 | 0.91 |
| 192.168.220.6 | 0.82 | 0.93 | 0.82 | 0.99 |
| 192.168.220.7 | 0.24 | 0.81 | 0.53 | 0.96 |
| 192.168.220.8 | 0.63 | 0.54 | 0.30 | 0.96 |
| 192.168.220.9 | 0.83 | 0.85 | 0.31 | 0.97 |
| 192.168.220.10 | 0.86 | 0.64 | 0.52 | 0.98 |
| 192.168.220.11 | 0.83 | 0.32 | 0.48 | 0.98 |
| 192.168.220.12 | 0.51 | 0.97 | 0.41 | 0.96 |
| 192.168.220.13 | 0.92 | 0.86 | 0.22 | 0.96 |
| 192.168.220.14 | 0.68 | 0.37 | 0.34 | 0.99 |

displays the p-value of each client. To further evaluate the performance of our model, we plot the number of packets, bytes, and duration of packets each client sent during a week. To make it easier to display, we select and plot clients by the p-value from Table 5.3. We select clients 192.168.220.7, 192.220.10, and 192.168.220.13 for displays in Figure 5.3. From the table, the p-value of client 192.168.220.10 indicate overall high similarity of real data and generated data. For client 192.168.220.7, we observe there are large and rare peaks in the real data, which makes it difficult for the GAN model to learn behavior. Despite this, for clients such as 192.168.220.13 that have more regular and predictable behaviors of bytes and packets sent to the server, the performance of our model indicated by p-value and plot is favorable. We evaluate the performance of the File Server by the same method. Table 5.3 shows the t test results for the File Server (192.168.100.5).

We observe that the attributes of packets sent by file server are usually fixed; for instance, the *bytes* attribute in the real data of the server is always 108. Thus, we perform the t test on *Date First Seen*, and the frequency of packets sent by File Server to verify the behavior of the server is captured by our model. Table 5.3 displays that the p-values are larger than 0.90.

## 5.6 Conclusion

We propose a defensive deception structure, named GanFlow, to assist honeypots in mimicking real nodes' network patterns, such as the number of packets in the network flow. We adopt a synthetic flow-based network traffic generator model based on improved WGAN. Flow-based traffic consists of heterogeneous data, but GANs can only take continuous data. To overcome this, we adopt deep learning approaches to handle flow-based traffic. We use t-test and distribution of generated and real client's network data to verify that, our deception model can effectively imitate the actual hosts' network flow. Our research indicates that deep learning approaches and GAN are capable of assisting honeypots in the defensive deception structure.

This research creates a GAN-based model to mimic the network flows. However, we have not implemented and analyzed our model with a realistic network environment. The future work includes deploying the deception structure while considering the actual network. In addition, a generated GanFlow may confuse attackers, but excessive GanFlows can disturb legitimate users or network functions, such as causing traffic congestion. Therefore, another future work is dynamic analyzing network situations and assisting the defender in searching for the optimal strategy to apply a GanFlow.

CHAPTER

6

# CONCLUSIONS AND FUTURE DIRECTIONS

This dissertation aims to explore and improve defensive deception through the use of game theory and machine learning. Our goal is to help the defender protect their assets from malicious reconnaissance and insider attackers.

**Question 1:** *how should the defender improve its effectiveness in enticing the attacker through deception?*

We present Mee , HoneyMee (Chapter 3), and GanFlow (Chapter 5) as solutions to Question 1. In Mee and HoneyMee, we use file sensitivity to measure the appeal of honeyfiles to both malicious insiders and legitimate users. This parameter is used to help the defender analyze honeyfile alarms and evaluate the security of the system. Furthermore, Mee can adjust the number of honeyfiles to help the defender increase the number of honeyfiles in high-security level devices, thus drawing increased attention from malicious insiders and being better able to detect the attacker and their intentions. The results in Figure 3.4 show that different numbers of honeyfiles may have different effectiveness in detecting attackers or draw more attention from attackers to cover real files. In our GanFlow research, we use a GAN to identify the patterns of legitimate servers and hosts. By utilizing these patterns, honeypots can mimic the corresponding device and draw the attention of attackers when performing malicious passive monitoring.

**Question 2:** *how should the defender effectively allocate resources?*

Mee, HoneyMee (Chapter 3), and hypergame-based hybrid honeypots (Chapter 4) are the solutions to Question 2. Recalling the groups in the Mee and HoneyMee approaches, we employ groups and their associated group values to differentiate the importance of devices to the defender. The number of honeyfiles that can be deployed on devices can be adjusted based on the groups' values and the system's security level, thus preventing unnecessary costs. Furthermore, by increasing or decreasing the number of honeyfiles on devices, the detection rate for insider attackers can be improved while the impact on legitimate users can be minimized.

In hybrid honeypots based on hypergames research, we use hypergames to assist the defender in allocating the resources to low and high-interaction honeypots. High-interaction honeypots offer greater performance and believability than low-interaction honeypots, but the cost of deploying them is higher. The hypergame scheme helps the defender find the most suitable strategy for allocating resources between low and high-interaction honeypots.

**Question 3:** *how should the defender reduce the impact of deception methods (e.g., confusing legitimate users)?*

We address this question by Mee and HoneyMee (Chapter 3). Both of these techniques take into account legitimate users as part of the environment. We imitate a user's actions in the simulations. To gain a better comprehension of the user's habits and enhance the realism of attackers, users, and the environment, we extend Mee to HoneyMee with DRL.

In all studies conducted for this dissertation, defenders must take into account the expense of implementing defensive deception when formulating strategies. They should adjust their strategies accordingly to maximize efficiency.

## Future Research Directions

This research reveals some interesting research directions, of which we find the following attractive:

- **Better metrics to measure the quality of honey-X technologies**: The quality of honey-X in our study has been mainly assessed based on its detection accuracy and players' payoff. There should be a wider range of metrics to measure the roles of both protecting assets and detecting attacks. Potentially, such metrics could estimate the level of uncertainty experienced by attackers, the number of vulnerable assets that attackers have failed to identify, the number of novel attack vectors collected through honeypots, and the number of important system components that have been attacked.

- **Measure the quality of defensive deception**: The quality of a DD technique should be determined based on how well it deceives attackers. That is, the quality of DD should be measured by the attacker's view and actions based on its belief about the defender's moves. However, existing work mainly uses system metrics as a proxy to measure the quality of a DD technique deployed by the defender. Some example metrics are the degree of uncertainty perceived by attackers, the degree of the discrepancy between the attacker's perception of the defense system and the ground truth system states, and service availability without being disrupted by deployed DD techniques.

- **Build a realistic testbed to evaluate defensive deception techniques based on game theory and machine learning**: Such approaches have been predominantly studied through mathematical proof or simulation models. To validate these approaches in a more realistic manner, emulation or real testbeds should be used, taking into account intelligent attackers and highly uncertain network environments. It is essential to consider how much and how accurate information is available to players, as well as how their beliefs are formulated in real settings. This is due to the fact that strong assumptions on common knowledge in game theory cannot be applied in real environments. To address this issue, the first step is to create a game where two players may have asynchronous information with imperfect or incomplete information, such as a hypergame [Bakker et al., 2021].

- **Development of defensive deception technologies based on cooperation between industry and academia:** The academic research community has widely discussed defensive deception technologies. Cybersecurity companies have created defensive deception products to detect or reduce threats, mainly honeypots. These security products mainly focus on counteracting ransomware, credential theft, or insider attack, and are mainly designed for enterprise networks or cloud-based networks. Unfortunately, since the companies have not revealed the core technologies used in their products, it is impossible to determine if game theory or machine learning approaches have been employed. Therefore, it is essential for industry and academia to work together to create more advanced defensive deception technologies based on the latest game theory and machine learning approaches.

References

Nazmiye Ceren Abay, Cuneyt Gurcan Akcora, Yan Zhou, Murat Kantarcioglu, and Bhavani Thuraisingham. Using deep learning to generate relational HoneyData. In *Autonomous Cyber Deception*, pages 3–19. Springer, 2019.

Stefan Achleitner, Thomas La Porta, Patrick McDaniel, Shridatt Sugrim, Srikanth V. Krishnamurthy, and Ritu Chadha. Cyber deception: Virtual networks to defend insider reconnaissance. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*, pages 57–68, 2016.

Palvi Aggarwal, Cleotilde Gonzalez, and Varun Dutt. Cyber-security: Role of deception in cyber-attack detection. In *Advances in Human Factors in Cybersecurity*, pages 85–96. Springer, 2016.

Palvi Aggarwal, Cleotilde Gonzalez, and Varun Dutt. Modeling the effects of amount and timing of deception in simulated network scenarios. In *Proc. 2017 Int'l Conf. On Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA)*, pages 1–7. IEEE, 2017.

Palvi Aggarwal, Aksh Gautam, Vaibhav Agarwal, Cleotilde Gonzalez, and Varun Dutt. HackIt: A human-in-the-loop simulation tool for realistic cyber deception experiments. In *Proc. Int'l Conf. on Applied Human Factors and Ergonomics*, pages 109–121. Springer, 2019.

Md Al Amin, Sachin Shetty, Laurent Njilla, Deepak Tosh, and Charles Kamouha. Attacker capability based dynamic deception model for large-scale networks. *EAI Endorsed Transactions on Security and Safety*, 6(21), 2019a.

Md Ali Reza Al Amin, Sachin Shetty, Laurent Njilla, Deepak K Tosh, and Charles Kamhoua. Online cyber deception system using partially observable Monte-Carlo planning framework. In *Proc. Int'l Conf. on Security and Privacy in Communication Systems*, pages 205–223. Springer, 2019b.

Md Ali Reza Al Amin, Sachin Shetty, Laurent L Njilla, Deepak K Tosh, and Charles A Kamhoua. Dynamic cyber deception using partially observable monte-carlo planning framework. *Modeling and Design of Secure Internet of Things*, pages 331–355, 2020.

Ehab Al-Shaer, Jinpeng Wei, Kevin W. Hamlen, and Cliff Wang. Towards intelligent cyber deception systems. In *Autonomous Cyber Deception*, pages 21–33. Springer, 2019.

Mohammed H Almeshekah and Eugene H Spafford. Planning and integrating deception into computer security defenses. In *Proceedings of the New Security Paradigms Workshop*, pages 127–138, 2014.

Mohammed H Almeshekah and Eugene H Spafford. Cyber security deception. In *Cyber Deception*, pages 23–50. Springer, 2016.

S. Almotairi, A. Clark, G. Mohay, and J. Zimmermann. A technique for detecting new attacks in low-interaction honeypot traffic. In *2009 Fourth International Conference on Internet Monitoring and Protection*, pages 7–13, 2009. doi: 10.1109/ICIMP.2009.9.

A. Alshammari, D. B. Rawat, M. Garuba, C. A. Kamhoua, and L. L. Njilla. *Deception for Cyber Adversaries: Status, Challenges, and Perspectives*, pages 141–160. Wiley Online Library, 2020.

Sara A Althubiti, Eric Marcell Jones, and Kaushik Roy. Lstm for anomaly-based network intrusion detection. In *2018 28th International telecommunication networks and applications conference (ITNAC)*, pages 1–3. IEEE, 2018.

Iffat Anjum, Mohammad Sujan Miah, Mu Zhu, Nazia Sharmin, Christopher Kiekintveld, William Enck, and Munindar P Singh. Optimizing vulnerability-driven honey traffic using game theory. *arXiv preprint arXiv:2002.09069*, 2020.

Iffat Anjum, Mu Zhu, Isaac Polinsky, William H. Enck, Michael K. Reiter, and Munindar P. Singh. Role-based deception in enterprise networks. In *Proceedings of the 11th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 65–76, Online, April 2021. ACM. doi: 10.1145/3422337.3447824.

A. H. Anwar, C. Kamhoua, and N. Leslie. Honeypot allocation over attack graphs in cyber deception games. In *Proc. 2020 Int'l Conf. on Computing, Networking and Communications (ICNC)*, pages 502–506, 2020.

Ahmed H Anwar and Charles Kamhoua. Game theory on attack graph for cyber deception. In *Proc. Int'l Conf. on Decision and Game Theory for Security*. Springer, 2020.

Ahmed H Anwar, Charles Kamhoua, and Nandi Leslie. A game-theoretic framework for dynamic cyber deception in Internet of Battlefield Things. In *Proc. 16th EAI Int'l Conf. on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 522–526, 2019.

Frederico Araujo, Kevin W. Hamlen, Sebastian Biedermann, and Stefan Katzenbeisser. From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation. In *Proc. ACM SIGSAC Conf. on Computer and Communications Security*, pages 942–953, New York, NY, USA, 2014. ACM.

C. A. Ardagna, M. Cremonini, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. Location privacy protection through obfuscation-based techniques. In Steve Barker and Gail-Joon Ahn, editors, *Proc. Data and Applications Security XXI*, pages 47–60, Berlin, 2007. Springer.

Ofir Arkin and Fyodor Yarochkin. Xprobe v2. 0. *Fuzzy" Approach to Remote Active Operating Systems Fingerprinting," http://www. xprobe2. org*, 2002.

Hassan Artail, Haidar Safa, Malek Sraj, Iyad Kuwatly, and Zaid Al-Masri. A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks. *computers & security*, 25(4):274–288, 2006.

Jeffrey Avery and Eugene H. Spafford. Ghost patches: Fake patches for fake vulnerabilities. In Sabrina De Capitani di Vimercati and Fabio Martinelli, editors, *Proc. 32nd IFIP TC 11 Int'l Conf. on ICT Systems Security and Privacy Protection*, pages 399–412, Rome, 2017. Springer Int'l Publishing.

Jeffrey Avery and John Ross Wallrabenstein. Formally modeling deceptive patches using a game-based approach. *Computers & Security*, 75:182–190, 2018.

Prudhvi Ratna Badri Satya, Kyumin Lee, Dongwon Lee, Thanh Tran, and Jason Jiasheng Zhang. Uncovering fake likers in online social networks. In *Proc. 25th ACM Int'l on Conf. on Information and Knowledge Management*, pages 2365–2370. ACM, 2016.

Craig Bakker, Arnab Bhattacharya, Samrat Chatterjee, and Draguna L Vrabie. Metagames and hypergames for deception-robust control. *ACM Transactions on Cyber-Physical Systems*, 5(3):1–25, 2021.

Kiran Bandla. Aptnotes. https://github.com/kbandla/APTnotes.git, 2019.

Gunjan Bansal, Niteesh Kumar, Sukumar Nandi, and Santosh Biswas. Detection of NDP based attacks using MLD. In *Proceedings of the Fifth International Conference on Security of Information and Networks*, pages 163–167, 2012.

Genevieve Bartlett, John Heidemann, and Christos Papadopoulos. Understanding passive and active service discovery. In *Proc. 7th ACM SIGCOMM Conf. on Internet measurement*, pages 57–70, 2007.

Anjon Basak, Charles Kamhoua, Sridhar Venkatesan, Marcus Gutierrez, Ahmed H. Anwar, and Christopher Kiekintveld. Identifying stealthy attackers in a game theoretic framework using deception. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 21–32. Springer, 2019.

Tamer Başar and Geert Jan Olsder. *Dynamic Noncooperative Game Theory*, volume 23. Siam, 1999.

J. Bowyer Bell and Barton Whaley. *Cheating and Deception.* Transaction, New York, 1991.

David Bellhouse. The problem of Waldegrave. *Electronic Journal for the History of Probability and Statistics*, 3(2):1–12, 2007.

Malek Ben Salem and Salvatore Stolfo. Combining baiting and user search profiling techniques for masquerade detection. *Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications*, 3(1), March 2012.

Michael Bennett and Edward Waltz. *Counterdeception Principles and Applications for National Security.* Artech House, 2007.

Kevin Benton, L Jean Camp, and Chris Small. Openflow vulnerability assessment. In *Proc. 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 151–152, 2013.

Mark Bilinski, Kimberly Ferguson-Walter, Sunny Fugate, Ryan Gabrys, Justin Mauger, and Brian Souza. You only lie twice: A multi-round cyber deception game of questionable veracity. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 65–84. Springer, 2019.

Jean-Marie Borello and Ludovic Mé. Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology*, 4(3):211–220, 2008.

Brian M Bowen, Shlomo Hershkop, Angelos D Keromytis, and Salvatore J Stolfo. Baiting inside attackers using decoy documents. In *Proc. Int'l Conf. on Security and Privacy in Communication Systems*, pages 51–70. Springer, 2009.

Cristian Cadar, Daniel Dunbar, and Dawson R Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Operating Systems Design and Implementation*, volume 8, pages 209–224, 2008.

Joseph W. Caddell. Deception 101-primer on deception. Technical report, DTIC Document, 2004.

Jin-Yi Cai, Vinod Yegneswaran, Chris Alfeld, and Paul Barford. An attacker-defender game for honeynets. In *Proc. Int'l Computing and Combinatorics Conf.*, pages 7–16. Springer, 2009.

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

William Casey, Jose Andre Morales, Evan Wright, Quanyan Zhu, and Bud Mishra. Compliance signaling games: Toward modeling the deterrence of insider threats. *Computational and Mathematical Organization Theory*, 22(3):318–349, 2016.

William Austin Casey, Quanyan Zhu, Jose Andre Morales, and Bud Mishra. Compliance control: Managed vulnerability surface in social-technological systems via signaling games. In *Proc. 7th ACM CCS Int'l Workshop on Managing Insider Security Threats*, pages 53–62, 2015.

Anthony R Cassandra. A survey of POMDP applications. In *Proc. Working notes of AAAI 1998 fall Symp. on planning with partially observable Markov decision processes*, volume 1724, 1998.

Hayreddin Çeker, Jun Zhuang, Shambhu Upadhyaya, Quang Duy La, and Boon-Hee Soong. Deception-based game theoretical approach to mitigate DoS attacks. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 18–38. Springer, 2016.

Ritu Chadha, Thomas Bowen, Cho-Yu J Chiang, Yitzchak M Gottlieb, Alex Poylisher, Angello Sapello, Constantin Serban, Shridatt Sugrim, Gary Walther, Lisa M Marvel, et al. CyberVAN: A cyber security virtual assured network testbed. In *Proc. MILCOM 2016-2016 IEEE Military Communications Conf.*, pages 1125–1130. IEEE, 2016.

Jien-Tsai Chan and Wuu Yang. Advanced obfuscation techniques for Java bytecode. *Journal of Systems and Software*, 71(1):1–10, April 2004.

Axel Charpentier, Nora Boulahia Cuppens, Frédéric Cuppens, and Reda Yaich. Deep reinforcement learning-based defense strategy selection. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pages 1–11, 2022.

Ping Chen, Lieven Desmet, and Christophe Huygens. A study on Advanced Persistent Threats. In *Proc. IFIP Int'l Conf. on Communications and Multimedia Security*, pages 63–72. Springer, 2014.

Adriel Cheng. PAC-GAN: Packet generation of network traffic using generative adversarial networks. In *Proceeding of the IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0728–0734. IEEE, 2019.

Cho-Yu J Chiang, Yitzchak M Gottlieb, Shridatt James Sugrim, Ritu Chadha, Constantin Serban, Alex Poylisher, Lisa M Marvel, and Jonathan Santos. ACyDS: An adaptive cyber deception system. In *Proc. 2016 IEEE Military Communications Conf.*, pages 800–805. IEEE, 2016.

Cho-Yu J Chiang, Sridhar Venkatesan, Shridatt Sugrim, Jason A. Youzwak, Ritu Chadha, Edward I. Colbert, Hasan Cam, and Massimiliano Albanese. On defensive cyber deception: A case study using SDN. In *Proc. IEEE Military Communications Conf. (MILCOM)*, pages 110–115. IEEE, 2018.

J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Communications Surveys Tutorials*, pages 1–1, 2020. Early access.

Jin-Hee Cho and Noam Ben-Asher. Cyber defense in breadth: Modeling and analysis of integrated defense systems. *The Journal of Defense Modeling and Simulation*, 15(2):147–160, 2018.

Jin-Hee Cho, Mu Zhu, and Munindar P. Singh. Modeling and analysis of deception games based on hypergame theory. In *Autonomous Cyber Deception*, pages 49–74. Springer, 2019.

Andrew Clark, Quanyan Zhu, Radha Poovendran, and Tamer Başar. Deceptive routing in relay networks. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 171–185. Springer, 2012.

Richard Colbaugh and Kristin Glass. Proactive defense for evolving cyber threats. In *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics*, pages 125–130. IEEE, 2011.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

Edward Cranford, Cleotilde Gonzalez, Palvi Aggarwal, Sarah Cooney, Milind Tambe, and Christian Lebiere. Adaptive cyber deception: Cognitively informed signaling for cyber defense. In *Proc. 53rd Hawaii Int'l Conf. on System Sciences*, 2020.

Edward A Cranford, Cleotilde Gonzalez, Palvi Aggarwal, S Cooney, M Tambe, and C Lebiere. Towards personalized deceptive signaling for cyber defense using cognitive models. In *Proc. 17th Annual Meeting of the ICCM, Montreal, CA*, volume 56, 2019.

Michael Crouse, Bryan Prosser, and Errin W Fulp. Probabilistic performance analysis of moving target and deception reconnaissance defenses. In *Proc. 2nd ACM Workshop on Moving Target Defense*, pages 21–29, 2015.

Donald Charles Daniel and Katherine Lydigsen Herbig. *Strategic Military Deception*. Pergamon, 1982.

P. Danzig, J. Mogul, V. Paxson, and M. Schwartz. The Internet traffic archive, 2008. URL http://ita.ee.lbl.gov/html/traces.html.

Prithviraj Dasgupta and Joseph Collins. A survey of game theoretic approaches for adversarial machine learning in cybersecurity tasks. *AI Magazine*, 40(2):31–43, June 2019.

Eddie Dekel, Drew Fudenberg, and David K Levine. Learning to play bayesian games. *Games and Economic Behavior*, 46(2):282–303, 2004.

Christos K Dimitriadis. Improving mobile core network security with honeynets. *IEEE Security & Privacy*, 5(4):40–47, 2007.

Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, and Ramana Kompella. Towards an elastic distributed SDN controller. In *Proc. 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 7–12, 2013.

Basirudin Djamaluddin, Ahmed Alnazeer, and Farag Azzedin. Web deception towards moving target defense. In *Proc. 2018 Int'l Carnahan Conf. on Security Technology (ICCST)*, pages 1–5. IEEE, 2018.

Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2):103–130, 1997.

David Donahue and Anna Rumshisky. Adversarial text generation without reinforcement learning. *ArXiv*, abs/1810.06640, 2018.

J. F. Dunnigan and A. A. Nofi. *Victory and deceit*. Writers Club Press, 2nd edition, 2001.

K. Durkota, V. Lisỳ, B. Bošanskỳ, and C. Kiekintveld. Optimal network security hardening using attack graph games. In *Proc. 24th Int'l Joint Conf. on Artificial Intelligence*, 2015.

Kevin P Dyer, Scott E Coull, and Thomas Shrimpton. Marionette: A programmable network traffic obfuscation system. In *24th USENIX Security Symposium ({USENIX} Security 15)*, pages 367–382, 2015.

A. I. M. Efendi, Z. Ibrahim, M. N. A. Zawawi, F. Abdul Rahim, N. A. M. Pahri, and A. Ismail. A survey on deception techniques for securing web application. In *Proc. 2019 IEEE 5th Int'l Conf. on Big Data Security on Cloud (BigDataSecurity)*, pages 328–331, 2019.

E. B. El Idrissi Younes, E. M. Fatna, and M. Nisrine. A security approach for social networks based on honeypots. In *Proc. 2016 4th IEEE Int'l Colloquium on Information Science and Technology (CiSt)*, pages 638–643, 2016.

A. El-Kosairy and M. A. Azer. A new web deception system framework. In *Proc. 1st Int'l Conf. on Computer Applications & Information Security (ICCAIS)*, pages 1–10. IEEE, 2018.

K. Ferguson-Walter, T. Shade, A. Rogers, M. C. S. Trumbo, K. S Nauer, K. M. Divis, A. Jones, A. Combs, and R. G. Abbott. The tularosa study: An experimental design and implementation to quantify the effectiveness of cyber deception. Technical report, Sandia National Lab (SNL-NM), Albuquerque, NM (United States), 2018.

K. Ferguson-Walter, S. Fugate, J. Mauger, and M. Major. Game theory for adaptive defensive cyber deception. In *Proc. 6th Annual Symp. on Hot Topics in the Science of Security*, page 4. ACM, 2019a.

K. Ferguson-Walter, S. Fugate, J. Mauger, and M. Major. Game theory for adaptive defensive cyber deception. In *Proc. 6th Annual Symp. on Hot Topics in the Science of Security*, pages 1–8, 2019b.

K. J. Ferguson-Walter. *An empirical assessment of the effectiveness of deception for cyber defense*. PhD thesis, University of Massachusetts Amherst, 2020.

Michelle Forelle, Phil Howard, Andrés Monroy-Hernández, and Saiph Savage. Political bots and the manipulation of public opinion in venezuela. *arXiv Preprint arXiv:1507.07109*, 2015.

N. M. Fraser and K. W. Hipel. *Conflict Analysis: Models and Resolutions*. North-Holland, 1984.

Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29 (2):131–163, 1997.

Xinwen Fu, Bryan Graham, Dong Xuan, Riccardo Bettati, and Wei Zhao. Empirical and theoretical evaluation of active probing attacks and their countermeasures. In *International Workshop on Information Hiding*, pages 266–281. Springer, 2004.

Nandan Garg and Daniel Grosu. Deception in honeynets: A game-theoretic analysis. In *Proc. IEEE SMC Information Assurance and Security Workshop*, pages 107–113. IEEE, 2007.

Piotr J Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

J. A. Gómez-Hernández, L. Álvarez-González, and P. García-Teodoro. R-Locker: Thwarting ransomware action through a honeyfile-based approach. *Computers & Security*, 73:389–398, 2018.

Ian Goodfellow. NIPS 2016 tutorial: Generative Adversarial Networks. *arXiv preprint arXiv:1701.00160*, 2016.

Priyanka Goyal, Sahil Batra, and Ajit Singh. A literature review of security attack in mobile ad-hoc networks. *International Journal of Computer Applications*, 9(12):11–15, 2010.

Alonso Granados, Mohammad Sujan Miah, Anthony Ortiz, and Christopher Kiekintveld. A realistic approach for network traffic obfuscation using adversarial machine learning. In *Proc. International Conference on Decision and Game Theory for Security*, pages 45–57. Springer, 2020.

A Greenwald. Matrix games and nash equilibrium. *Lecture in Game-theoretic Artificial Intelligence, Brown University CS Dep. Providence, US*, 2007.

K. Grover, A. Lim, and Q. Yang. Jamming and anti–jamming techniques in wireless networks: A survey. *Int'l Journal of Ad Hoc and Ubiquitous Computing*, 17(4):197–215, 2014.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838. PMLR, 2016.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5769–5779, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Z. Guo, J.-H. Cho, I.-R. Chen, S. Sengupta, M. Hong, and T. Mitra. Online social deception and its countermeasures for trustworthy cyberspace: A survey, 2020a.

Zhen Guo, Jin-Hee Cho, Ray Chen, Srijan Sengupta, Michin Hong, and Tanushree Mitra. Online social deception and its countermeasures: A survey. *IEEE Access*, 2020b.

J. Han, J. Pei, and M. Kamber. *Data Mining: Concepts and Techniques.* Elsevier, 2011.

X. Han, N. Kheir, and D. Balzarotti. Deception techniques in computer security: A research perspective. *ACM Computing Surveys*, 51(4), July 2018.

E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715, 2004.

John C. Harsanyi. Games with incomplete information played by "Bayesian" players, I–III Part I. the basic model. *Management Science*, 14(3):159–182, 1967.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

J. P. Hespanha, Y. S. Ateskan, H. Kizilocak, et al. Deception in non-cooperative games with partial information. In *Proc. 2nd DARPA-JFACC Symp. on Advances in Enterprise Control*, pages 1–9, 2000.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, and Ian Osband. Deep Q-learning from demonstrations. In *Thirty-second AAAI Conference on Artificial Intelligence*, 2018.

W. Hofer, T. Edgar, D. Vrabie, and K. Nowak. Model-driven deception for control system environments. In *Proc. IEEE Int'l Symp. on Technologies for Homeland Security (HST)*, pages 1–7. IEEE, 2019.

K. Horák, Q. Zhu, and B. Bošanskỳ. Manipulating adversary's belief: A dynamic game approach to deception by design for proactive network security. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 273–294. Springer, 2017.

James Thomas House and George Cybenko. Hypergame theory applied to cyber attack and defense. In *Proc. SPIE Conf. on Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense IX*, volume 766604, May 2010.

L. Huang and Q. Zhu. Dynamic Bayesian games for adversarial and defensive cyber deception. In *Autonomous Cyber Deception*, pages 75–97. Springer, 2019.

Jafar Haadi Jafarian and Amirreza Niakanlahiji. A deception planning framework for cyber defense. In *Proc. 53rd Hawaii Int'l Conf. on System Sciences*, 2020.

Sushil Jajodia, Noseong Park, Fabio Pierazzi, Andrea Pugliese, Edoardo Serra, Gerardo I. Simari, and V. S. Subrahmanian. A probabilistic logic of cyber deception. *IEEE Transactions on Information Forensics and Security*, 12(11):2532–2544, 2017.

A. Jøsang, J. Cho, and F. Chen. Uncertainty characteristics of subjective opinions. In *Proc. 2018 21st Int'l Conf. on Information Fusion (FUSION)*, pages 1998–2005, 2018.

Atsushi Kajii and Stephen Morris. The robustness of equilibria to incomplete information. *Econometrica: Journal of the Econometric Society*, pages 1283–1309, 1997.

C. A. Kamhoua. Game theoretic modeling of cyber deception in the Internet of Battlefield Things. In *Proc. 2018 56th Annual Allerton Conf. on Communication, Control, and Computing (Allerton)*, pages 862–862. IEEE, 2018.

C.A. Kamhoua, C.D. Kiekintveld, F. Fang, and Q. Zhu. *Game Theory and Machine Learning for Cyber Security.* Wiley, 2021.

S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. *SIGCOMM Comput. Commun. Rev.*, 39(4):243–254, August 2009.

Pradeeban Kathiravelu and Luís Veiga. Sd-cps: taming the challenges of cyber-physical systems with a software-defined approach. In *Proc. 2017 Fourth Int'l Conf. on Software Defined Systems (SDS)*, pages 6–13. IEEE, 2017.

A. D. Keromytis and S. J. Stolfo. Systems, methods, and media for generating bait information for trap-based defenses, August 2014. US Patent 8,819,825.

C. Kiekintveld, V. Lisỳ, and R. Píbil. Game-theoretic foundations for the strategic use of honeypots in network security. In *Cyber Warfare*, pages 81–101. Springer, 2015.

Tae Kyun Kim. T test as a parametric statistic. *Korean journal of anesthesiology*, 68(6):540, 2015.

M. Kocakulak and I. Butun. An overview of wireless sensor networks towards Internet of Things. In *Proc. IEEE 7th Annual Computing and Communication Workshop and Conf. (CCWC)*, pages 1–6. IEEE, 2017.

A. Kott, A. Swami, and B. West. The Internet of Battle Things. *IEEE Computer*, December 2016.

T. Krueger, H. Gascon, N. Krämer, and K. Rieck. Learning stateful models for network honeypots. In *Proc. 5th ACM workshop on Security and artificial intelligence*, pages 37–48, 2012.

Abhishek N. Kulkarni, Jie Fuand Huan Luo, Charles A. Kamhoua, and Nandi N. Leslie. Decoy allocation games on graphs with temporal logic objectives. In *Proc. Int'l Conf. on Decision and Game Theory for Security.* Springer, 2020.

Abhishek N Kulkarni, Huan Luo, Nandi O Leslie, Charles A Kamhoua, and Jie Fu. Deceptive labeling: Hypergames on graphs for stealthy deception. *IEEE Control Systems Letters*, July, 2021.

Q. D. La, T. Q. Quek, J. Lee, S. Jin, and H. Zhu. Deceptive attack and defense game in honeypot-enabled networks for the Internet-of-Things. *IEEE Internet of Things Journal*, 3(6):1025–1035, 2016.

K. Lee, J. Caverlee, and S. Webb. Uncovering social spammers: Social honeypots + machine learning. In *Proc. 33rd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 435–442, 2010.

K. Lee, B. D. Eoff, and J. Caverlee. Seven months with the devils: A long-term study of content polluters on Twitter. In *Proc. 5th Int'l AAAI Conf. on Weblogs and Social Media*, pages 185–192, 2011.

Huanruo Li, Yunfei Guo, Shumin Huo, Hongchao Hu, and Penghao Sun. Defensive deception framework against reconnaissance attacks in the cloud with deep reinforcement learning. *Science China Information Sciences*, 65(7):170305, 2022.

Fang Liu, Xiuzhen Cheng, and Dechang Chen. Insider attacker detection in wireless sensor networks. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 1937–1945. IEEE, 2007.

Z. Lu, C. Wang, and S. Zhao. Cyber deception for computer and network security: Survey and challenges. *arXiv preprint arXiv:2007.14497*, 2020.

Thomas Lumley, Paula Diehr, Scott Emerson, and Lu Chen. The importance of the normality assumption in large public health data sets. *Annual review of public health*, 23:151–69, Feb. 2002. doi: 10.1146/annurev.publhealth.23.100901.140546.

Gordon Lyon. Nmap - Free Security Scanner For Network Exploration & Security Audits. https://nmap.org/, 2024. Accessed: 02-24-2024.

D. C. MacFarland and C. A. Shue. The SDN shuffle: Creating a moving-target defense using host-based software-defined networking. In *Proc. ACM Workshop on Moving Target Defense*, pages 37–41, 2015.

D. Mao, S. Zhang, L. Zhang, and Y. Feng. Game theory based dynamic defense mechanism for SDN. In *Proc. Int'l Conf. on Machine Learning for Cyber Security*, pages 290–303. Springer, 2019.

Mirco Marchetti, Fabio Pierazzi, Alessandro Guido, and Michele Colajanni. Countering Advanced Persistent Threats through security intelligence and big data analytics. In *2016 8th International Conference on Cyber Conflict (CyCon)*, pages 243–261. IEEE, 2016.

A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic theory*, volume 1. Oxford University Press New York, 1995.

Mohammad Sujan Miah, Mu Zhu, Alonso Granados, Nazia Sharmin, Iffat Anjum, Anthony Ortiz, Christopher Kiekintveld, William Enck, and Munindar P Singh. Optimizing honey traffic using game theory and adversarial learning. In *Cyber Deception: Techniques, Strategies, and Human Aspects*, pages 97–124. Springer, 2022.

Stephanie Milani, Weiran Shen, Kevin S. Chan, Sridhar Venkatesan, Nandi O. Leslie, Charles A. Kamhoua, and Fei Fang. Harnessing the power of deception in attack graph games. In *Proc. Int'l Conf. on Decision and Game Theory for Security.* Springer, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

A. Mohammadi, M. H. Manshaei, M. M. Moghaddam, and Q. Zhu. A game-theoretic analysis of deception over social networks using fake avatars. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 382–394. Springer, 2016.

D. Montigny-Leboeuf and F. Massicotte. Passive network discovery for real time situation awareness. Technical report, Communications Research Centre Ottawa (Ontario), 2004.

Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

S. Nan, S. Brahma, C. A. Kamhoua, and N. Leslie. Behavioral cyber deception: A game and prospect theoretic approach. In *Proc. 2019 IEEE Global Communications Conf. (GLOBECOM)*, pages 1–6. IEEE, 2019.

S. Nan, S. Brahma, C. A. Kamhoua, and L. L. Njilla. *On Development of a Game-Theoretic Model for Deception-Based Security*, pages 123–140. Wiley Online Library, 2020a.

Satyaki Nan, Swastik Brahma, Charles A Kamhoua, and Nandi O Leslie. Mitigation of jamming attacks via deception. In *Proc. 2020 IEEE 31st Annual Int'l Symp. on Personal, Indoor and Mobile Radio Communications*, pages 1–6. IEEE, 2020b.

S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang. Predicting network attack patterns in SDN using machine learning approach. In *Proc. IEEE Conf. on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 167–172. IEEE, 2016.

Jose Nazario. Phoneyc: A virtual client honeypot. *LEET*, 9:911–919, 2009.

Nextgate. Research report 2013 state of social media spam, 2019. URL https://www.slideshare.net/prayukth1/2013-state-of-social-media-spam-research-report. Accessed: 08-05-2019.

Thanh Thi Nguyen and Vijay Janapa Reddi. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3779–3795, 2023.

H. Okhravi, M. A. Rabe, W. G. Leonard, T. R. Hobson, D. Bigelow, and W. W. Streilein. Survey of cyber moving targets. TR 1166, Lexington Lincoln Lab, MIT, 2013.

Felix O. Olowononi, Danda B. Rawat, Charles A. Kamhoua, and Brian M. Sadler. Deep reinforcement learning for deception in IRS-assisted UAV communications. In *IEEE Military Communications Conference (MILCOM)*, pages 763–768, 2022.

N. Patki, R. Wedge, and K. Veeramachaneni. The synthetic data vault. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410, Oct 2016. doi: 10.1109/DSAA.2016.49.

J. Pawlick and Q. Zhu. Deception by design: Evidence-based signaling games for network defense. *arXiv preprint arXiv:1503.05458*, 2015.

J. Pawlick, S. Farhang, and Q. Zhu. Flip the cloud: Cyber-physical signaling games in the presence of Advanced Persistent Threats. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 289–308. Springer, 2015.

J. Pawlick, E. Colbert, and Q. Zhu. Modeling and analysis of leaky deception using signaling games with evidence. *IEEE Transactions on Information Forensics and Security*, 14(7):1871–1886, 2018.

J. Pawlick, E. Colbert, and Q. Zhu. A game-theoretic taxonomy and survey of defensive deception for cybersecurity and privacy. *ACM Computing Surveys (CSUR)*, 52(4):1–28, 2019.

T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys (CSUR)*, 39(1):3–es, 2007.

Luan Huy Pham, Massimiliano Albanese, Ritu Chadha, Cho-Yu J Chiang, Sridhar Venkatesan, Charles Kamhoua, and Nandi Leslie. A quantitative framework to model reconnaissance by stealthy attackers and support deception-based defenses. In *Proc. 2020 IEEE Conf. on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2020.

L. Phlips. *The economics of imperfect information.* Cambridge University Press, 1988.

R. Píbil, V. Lisỳ, C. Kiekintveld, B. Bošanskỳ, and M. Pěchouček. Game theoretic model of strategic honeypot selection in computer networks. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 201–220. Springer, 2012.

Antônio J Pinheiro, Jeandro M Bezerra, and Divanilson R Campelo. Packet padding for improving privacy in consumer iot. In *Proc. IEEE Symposium on Computers and Communications (ISCC)*, pages 00925–00929. IEEE, 2018.

S. Pinto, T. Gomes, J. Pereira, J. Cabral, and A. Tavares. Iioteed: An enhanced, trusted execution environment for industrial iot edge devices. *IEEE Internet Computing*, 21(1):40–47, 2017.

R. Poovendran. Cyber-Physical Systems: Close encounters between two parallel worlds. *Proc. IEEE*, 98 (8):1363–1366, August 2010.

J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

Mohammad Ashiqur Rahman, Mohammad Hossein Manshaei, and Ehab Al-Shaer. A game-theoretic approach for deceiving remote operating system fingerprinting. In *Proc. IEEE Conf. on Communications and Network Security (CNS)*, pages 73–81. IEEE, 2013.

Giorgia Ramponi, Pavlos Protopapas, Marco Brambilla, and Ryan Janssen. T-cgan: Conditional generative adversarial network for data augmentation in noisy time series with irregular sampling. *arXiv preprint arXiv:1811.08295*, 11 2018.

Shailendra Rathore, Pradip Kumar Sharma, Vincenzo Loia, Young-Sik Jeong, and Jong Hyuk Park. Social network security: Issues, challenges, threats, and solutions. *Information Sciences*, 421:43–69, 2017.

Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Creation of flow-based data sets for intrusion detection. *Journal of Information Warfare*, 16:40–53, 2017a.

Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Flow-based benchmark data sets for intrusion detection. In *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS)*, pages 361–369. ACPI, 2017b.

Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156–172, 2019. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2018.12.012.

J. J. P. C. Rodrigues, D. B. De Rezende Segundo, H. A. Junqueira, M. H. Sabino, R. M. Prince, J. Al-Muhtadi, and V. H. C. De Albuquerque. Enabling technologies for the Internet of Health Things. *IEEE Access*, 6:13129–13141, 2018.

Kohavi Ron and Provost Foster. Special issue on applications of machine learning and the knowledge discovery process. *Journal of Machine Learning*, 30:271–274, 1998.

Stephane Ross, Brahim Chaib-draa, and Joelle Pineau. Bayes-adaptive pomdps. In *NIPS*, pages 1225–1232, 2007.

Neil C Rowe. Finding logically consistent resource-deception plans for defense in cyberspace. In *Proc. 21st Int'l Conf. on Advanced Information Networking and Applications Workshops (AINAW'07)*, volume 1, pages 563–568. IEEE, 2007.

Neil C. Rowe and Julian Rrushi. *Introduction to Cyberdeception*. Springer, 2016.

Neil C Rowe, Binh T Duong, and E John Custy. Fake honeypots: A defensive tactic for cyberspace. In *Proc. IEEE Workshop on Information Assurance*, pages 223–230, 2006.

S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

Malek Ben Salem and Salvatore J Stolfo. Decoy document deployment for effective masquerade attack detection. In *Proc. Int'l Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 35–54. Springer, 2011.

Malek Ben Salem, Shlomo Hershkop, and Salvatore J Stolfo. A survey of insider attack detection research. In *Insider Attack and Cyber Security*, pages 69–90. Springer, 2008.

Muhammed O. Sayin and Tamer Başar. Deception-as-defense framework for cyber-physical systems. *arXiv preprint arXiv:1902.01364*, 2019.

Aaron Schlenker, Omkar Thakoor, Haifeng Xu, Fei Fang, Milind Tambe, and Phebe Vayanos. Game theoretic cyber deception to foil adversarial network reconnaissance. In *Adaptive Autonomous Secure Cyber Systems*, pages 183–204. Springer, 2020.

Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th USENIX Security Symposium (USENIX) Security 17)*, pages 1357–1374, 2017.

Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. SDN security: A survey. In *IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7. IEEE, 2013.

Christian Seifert, Ian Welch, Peter Komisarczuk, et al. Honeyc-the low-interaction client honeypot. *Proc. 2007 NZCSRCS, Waikato University, Hamilton, New Zealand*, 6, 2007.

D. Serpanos. The cyber-physical systems revolution. *Computer*, 51(3):70–73, 2018.

Jia Shan-Shan and Xu Ya-Bin. The APT detection method based on attack tree for SDN. In *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy*, pages 116–121, 2018.

Lloyd S. Shapley. Stochastic games. *Proc. National Academy of Sciences*, 39(10):1095–1100, 1953.

Walter L. Sharp. Military deception. Technical Report 3-13.4, Joint War-Fighting Center, Doctrine and Education Group, 2006.

Zheyuan R. Shi, Ariel D. Procaccia, Kevin S. Chan, Sridhar Venkatesan, Noam Ben-Asher, Nandi O. Leslie, Charles Kamhoua, and Fei Fang. Learning and planning in feature deception games. In *Proc. Int'l Conf. on Decision and Game Theory for Security*. Springer, 2020.

Reza Shokri. Privacy games: Optimal user-centric data obfuscation. *Proceedings on Privacy Enhancing Technologies*, 2015(2):299–315, 2015.

Chris Snijders, Uwe Matzat, and Ulf-Dietrich Reips. "big data": Big gaps of knowledge in the field of Internet science. *Int'l Journal of Internet Science*, 7(1):1–5, 2012.

Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation. In *Proc. First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 29–36, 2011.

Lance Spitzner. The honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 1(2):15–23, 2003.

Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proc. 26th Annual Computer Security Applications Conf.*, pages 1–9. ACM, 2010.

Steven Tadelis. *Game Theory: An Introduction.* Princeton University Press, 2013.

Hassan Takabi and J Haadi Jafarian. Insider threat mitigation using moving target defense and deception. In *Proc. 2017 Int'l Workshop on Managing Insider Security Threats*, pages 93–96, 2017.

Colin Tankard. Advanced Persistent Threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

Omkar Thakoor, Milind Tambe, Phebe Vayanos, Haifeng Xu, Christopher Kiekintveld, and Fei Fang. Cyber camouflage games for strategic deception. In *Proc. Int'l Conf. on Decision and Game Theory for Security*, pages 525–541. Springer, 2019.

Olivier Tsemogne, Yezekael Hayel, Charles A. Kamhoua, and Gabriel Deugoue. Partially observable stochastic games for cyber deception against epidemic process. In *Proc. Int'l Conf. on Decision and Game Theory for Security.* Springer, 2020.

R. Vane and P. E. Lehner. Using hypergames to select plans in adversarial environments. In *Proc. 1st Workshop on Game Theoretic and Decision Theoretic Agents*, pages 103–111, 1999.

Maria Vergelis, Tatyana Shcherbakova, and Tatyana Sidorina. Spam and phishing in Q1 2019, 2019. URL https://securelist.com/spam-and-phishing-in-q1-2019/90795/. Accessed: 05-19-2019.

Abhishek Verma and Virender Ranga. Statistical analysis of cidds-001 dataset for network intrusion detection systems using distance-based machine learning. *Procedia Computer Science*, 125:709–716, 2018.

Nikos Virvilis, Bart Vanautgaerden, and Oscar Serrano Serrano. Changing the game: The art of deceiving sophisticated attackers. In *Proc. 6th Int'l Conf. On Cyber Conflict (CyCon)*, pages 87–97. IEEE, 2014.

Heinrich Von Stackelberg. *Market structure and equilibrium.* Springer Science & Business Media, 2010.

Jonathan Voris, Jill Jermyn, Nathaniel Boggs, and Salvatore Stolfo. Fox in the trap: Thwarting masqueraders via automated decoy document deployment. In *Proceedings of the Eighth European Workshop on System Security*, pages 1–7, 2015.

Gérard Wagener, Alexandre Dulaunoy, Thomas Engel, et al. Self adaptive high interaction honeypots driven by game theory. In *Symp. on Self-Stabilizing Systems*, pages 741–755. Springer, 2009.

Omar Abdel Wahab, Jamal Bentahar, Hadi Otrok, and Azzam Mourad. I know you are watching me: Stackelberg-based adaptive intrusion detection strategy for insider attacks in the cloud. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 728–735. IEEE, 2017.

Zelin Wan, Jin-Hee Cho, Mu Zhu, Ahmed H Anwar, Charles Kamhoua, and Munindar P Singh. Foureye: Defensive deception against advanced persistent threats via hypergame theory. *IEEE Transactions on Network and Service Management*, 2021.

Gang Wang, Christo Wilson, Xiaohan Zhao, Yibo Zhu, Manish Mohanlal, Haitao Zheng, and Ben Y Zhao. Serf and turf: Crowdturfing for fun and profit. In *Proc. 21st Int'l Conf. on World Wide Web*, pages 679–688. ACM, 2012.

He Wang and Bin Wu. Sdn-based hybrid honeypot for attack capture. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1602–1606. IEEE, 2019.

Shuo Wang, Qingqi Pei, Jianhua Wang, Guangming Tang, Yuchen Zhang, and Xiaohu Liu. An intelligent deployment policy for deception resources based on reinforcement learning. *IEEE Access*, 8:35792–35804, 2020.

Bryan C Ward, Steven R Gomez, Richard Skowyra, David Bigelow, Jason Martin, James Landry, and Hamed Okhravi. Survey of cyber moving targets second edition. Technical report, MIT Lincoln Laboratory Lexington United States, 2018.

Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

Ben Whitham. Minimising paradoxes when employing honeyfiles to combat data theft in military networks. In *Proc. Military Communications and Information Systems Conf. (MilCIS)*, pages 1–6. IEEE, 2016.

Ben Whitham. Automating the generation of enticing text content for high-interaction honeyfiles. In *Proc. 50th Hawaii Int'l Conf. on System Sciences*, 2017.

Wikipedia contributors. K-means clustering — Wikipedia, the free encyclopedia, 2024. URL https://en.wikipedia.org/w/index.php?title=K-means_clustering&oldid=1209436636. [Online; accessed 27-February-2024].

B. Xi and C. A. Kamhoua. *A Hypergame-Based Defense Strategy Toward Cyber Deception in Internet of Battlefield Things (IoBT)*, pages 59–77. Wiley Online Library, 2020.

Jie Xu and Jun Zhuang. Modeling costly learning and counter-learning in a defender-attacker game with private defender information. *Annals of Operations Research*, 236(1):271–289, 2016.

W. Xu, F. Zhang, and S. Zhu. The power of obfuscation techniques in malicious JavaScript code: A measurement study. In *Proc. 7th Int'l Conf. on Malicious and Unwanted Software*, pages 9–16, 2012.

Chao Yang, Jialong Zhang, and Guofei Gu. A taste of tweets: Reverse engineering Twitter spammers. In *Proc. 30th Annual Computer Security Applications Conf.*, pages 86–95. ACM, 2014.

Yue Yin, Bo An, Yevgeniy Vorobeychik, and Jun Zhuang. Optimal deceptive strategies in security games: A preliminary study. In *Proc. AAAI Conf. on Artificial Intelligence*, 2013.

I. You and K. Yim. Malware obfuscation techniques: A brief survey. In *Proc. Int'l Conf. on Broadband, Wireless Computing, Communication and Applications*, pages 297–300, 2010.

J. J. Yuill. *Defensive Computer-Security Deception Operations: Processes, Principles and Techniques.* PhD thesis, North Carolina State University, 2006.

James Joseph Yuill. *Defensive computer-security deception operations: Processes, principles and techniques.* PhD thesis, North Carolina State University, 2007.

Jim Yuill, Mike Zappe, Dorothy Denning, and Fred Feer. Honeyfiles: Deceptive files for intrusion detection. In *Proc. 5th Annual IEEE Information Assurance Workshop*, pages 116–122, 2004.

Jim Yuill, Dorothy Denning, and Fred Feer. Using deception to hide things from hackers: Processes, principles, and techniques. *Journal of Information Warfare*, 5(3):26–40, 2006.

M. Zalewski. p0f v3, 2014. URL https://lcamtuf.coredump.cx/p0f3/. Accessed: 07-18-2021.

Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys & Tutorials*, 15 (4):2046–2069, 2013.

Xiaoxiong Zhang, Keith W. Hipel, Bingfeng Ge, and Yuejin Tan. A game-theoretic model for resource allocation with deception and defense efforts. *Systems Engineering*, 22(3):282–291, 2019.

Yan Zhou, Murat Kantarcioglu, and Bowei Xi. A survey of game theoretic approach for adversarial machine learning. *WIREs Data Mining and Knowledge Discovery*, 9(3):e1259, 2019.

Hangcheng Zhu. *Fighting Against Social Spammers on Twitter by Using Active Honeypots.* PhD thesis, McGill University Libraries, 2015.

Mu Zhu and Munindar P Singh. Mee: Adaptive honeyfile system for insider attacker detection. In *Cyber Deception: Techniques, Strategies, and Human Aspects*, pages 125–143. Springer, 2022.

Mu Zhu, Ahmed H Anwar, Zelin Wan, Jin-Hee Cho, Charles Kamhoua, and Munindar P Singh. Game-theoretic and machine learning-based approaches for defensive deception: A survey. *arXiv preprint arXiv:2101.10121*, 2021a.

Mu Zhu, Ahmed H Anwar, Zelin Wan, Jin-Hee Cho, Charles A Kamhoua, and Munindar P Singh. A survey of defensive deception: Approaches using game theory and machine learning. *IEEE Communications Surveys & Tutorials*, 23(4):2460–2493, 2021b.

Mu Zhu, Ruijie Xi, Nazia Sharmin, Mohammad Miah, Christopher Kiekintveld, and Munindar P. Singh. Honeyflow: Decoy network traffic generation via Generative Adversarial Network. In *IEEE Global Communications Conference: Communication & Information Systems Security*, Rio de Janeiro, Brazil, December 2022.

Quanyan Zhu, Andrew Clark, Radha Poovendran, and Tamer Başar. Deceptive routing games. In *Proc. 51st IEEE Conf. on Decision and Control (CDC)*, pages 2704–2711. IEEE, 2012.

Quanyan Zhu, Andrew Clark, Radha Poovendran, and Tamer Başar. Deployment and exploitation of deceptive honeybots in social networks. In *Proc. 52nd IEEE Conf. on Decision and Control*, pages 212–219. IEEE, 2013.