Perry: A What-If Analysis Platform for Deception Evaluation

Yusuf Bin Saquib

CMU-CS-23-132
August 2023

Carnegie Mellon University
Computer Science Department
School of Computer Science
Pittsburgh, PA, 15213

Thesis Committee

Vyas Sekar (Chair)
Lujo Bauer

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Keywords: Deception Evaluation, Cyber Deception, Deception Technology, Network Security, Information Security, Security and Privacy, Computer Networks, Cybersecurity

"اكره الخطأ دائما لكن لا تكره المخطئ... ابغض بكل قلبك المعصية لكن سامح وارحم العاصي... انتقد القول لكن احترم القائل... فإن مهمتنا أن نقضي على المرض لا على المريض"

إمام الشافعي

"Always hate what is wrong, but don't hate the one who errors. Hate sin with all your heart, but forgive and have mercy on the sinner. Criticize speech, but respect the speaker. Our job is to wipe out disease, not the patient."

Imam Al-Shafi'i

Abstract

As we become more reliant on resilient networks, it is increasingly imperative for cybersecurity researchers and professionals to refine their techniques against malicious attacks. Within the realm of network security, cyber deception emerges as a promising defensive technique to leverage the asymmetry between attackers and defenders. However, the lack of a standardized evaluation method makes evaluating the efficacy of deception techniques an arduous task. In this master's thesis, we present PERRY: a realistic, extensible, and automated platform that aims to evaluate the efficacy of various deception techniques via emulation and allows the user fine-grained control over all aspects of the platform.

We demonstrate the effectiveness of PERRY by using it to evaluate several defender profiles against an emulated attacker, running hundreds of trials and comparing the results. We found that allowing a defender to deploy deception techniques in addition to equipping it with telemetry prevents an attacker from completely succeeding in its goals over 80% of the time. Furthermore, employing smarter defender strategies that dynamically adapt to the attacker's actions allows the defender to prevent the attacker from completely succeeding in its goals nearly 95% of the time. Taking advantage of our platform's foundation, future researchers can build upon PERRY and extend it to realistically evaluate other deception techniques against various attackers and in a broad range of scenarios.

Acknowledgements

First and foremost, all praise and thanks are due to Allah ﷻ, the Most Gracious, the Most Merciful, for His blessings given to me during my studies and in completing this thesis, for without Him I would not have been able to do any of this. I ask Allah ﷻ that this knowledge benefits me and that I may benefit others with it.

I owe a profound debt of gratitude to my beloved parents. Their unwavering love, support, and encouragement are the bedrock upon which my life and academic pursuits have been built. Their selfless acts, from the frequent emotional check-ins to the groceries they sent me during my busiest days, are a testament to their love and care for me. I humbly acknowledge that I will never be able to repay them for all that they have done for me, and I consider it an honor to have been blessed with such loving parents. I pray that Allah ﷻ rewards them for their sacrifices and grants them the highest level of Jannah.

My heartfelt thanks go to my advisor, Vyas Sekar, whose guidance has been instrumental in my academic and personal growth, and to my co-advisor, Lujo Bauer, for his invaluable insights. Their combined wisdom, patience, and mentorship have been a beacon of light for me throughout this journey.

A special thanks to my friends and colleagues, Brian Singer and Trevor Kann, for their invaluable help and support with my thesis and for fostering a fun, welcoming environment in CyLab. Special recognition goes to Brian for naming our platform "Perry" and laying down its foundations.

I would also like to extend my sincere gratitude to Khaled Harras and Ryan Riley from Carnegie Mellon University Qatar. Their guidance from the very beginning and unwavering support throughout the application process to the fifth year master's program at Carnegie Mellon University has been pivotal in shaping my academic journey.

Last, but certainly not least, my siblings, grandparents, family, and friends deserve a world of thanks. Their love, encouragement, and presence, even from afar, have been a constant source of strength and motivation for me. I cherish them dearly and hold their support very close to my heart.

Contents

List of Figures

List of Tables

For my beloved parents…

Chapter 1

Introduction

In cybersecurity, there exists a large asymmetry between attackers and defenders; defenders must be perfect in protecting their assets against countless forms of threats ranging in sophistication (insiders, government agencies, hacker groups, script-kiddies, etc.), while the attacker only needs to find a miniscule gap in the defender's security with which it can gain access to the defender's resources and cause significant damage.

This asymmetry, however, is reversed in the real world. In traditional forms of warfare (namely not cyber warfare), defenders typically have a significant advantage over their attackers: the defenders have far better knowledge of their land than the attacker, they have mighty structures they can retreat to (castles, fortresses, towers, etc.), and the burden of advancing is largely on the attacker.

With the advent of cyber warfare, an interesting idea emerged: what if we could reverse the asymmetry in cybersecurity by leveraging real world military tactics? One such tactic is deception, as described by Sun Tzu in *The Art of War* [1].

> "*All warfare is based on deception. Hence, when we are able to attack, we must seem unable; when using our forces, we must appear inactive; when we are near, we must make the enemy believe we are far away; when far away, we must make him believe we are near.*"
>
> *Sun Tzu, The Art of War*

In the context of cybersecurity, deception is the set of deliberate actions taken to mislead attackers, thus causing them to take (or not take) specific actions that will benefit computer-security defenses. While deception has been discussed in the context of cybersecurity for several years, it has recently gained traction as a promising technique for defending against cyber attacks.

However, it is still unclear how effective deception really is at protecting networks from attackers. As such, in this thesis, we propose PERRY: a realistic, extensible, and automated platform that aims to evaluate the efficacy of various deception techniques via emulation and allows the user fine-grained control over the specific details of the platform.

1.1 Paper Outline

The rest of this paper is organized as follows: In Chapter 2, we present an overview of deception in the context of network security and several associated challenges. After that, in Chapter 3, we discuss the current state of research into the evaluation of deception techniques. In Chapter 4, we describe the abstract design of PERRY, which we then build upon in Chapter 5 by presenting the implementation details of PERRY. Following that, in Chapter 6 we test our system by conducting several experiments and discuss the results. We then discuss some limitations in Chapter 7 and future work in Chapter 8. Finally, we conclude our work in Chapter 9.

Chapter 2

Background

In this section, we present an overview of deception in the context of network security that is sufficient for the reader to understand the majority of this thesis. We will define cyber deception, discuss some use cases and applications, and explore some challenges associated with deception.

2.1 Cyber Deception

For the sake of brevity, we use the terms ”cyber deception” and ”deception” interchangeably.

2.1.1 What is Deception?

Deception is the act of deliberately misleading someone by concealing or presenting false information, typically in order to gain personal advantage. In the context of network security, **deception** is the set of deliberate actions taken to mislead attackers, thus causing them to take (or not take) specific actions that will benefit computer-security defenses [2]. A common example of a deception technique is the **honeypot**, which is a decoy system that is designed to appear identical to a real system [3], and attracts malicious actors to the valuable data it appears to contain. The purpose of honey-objects (decoy objects) is to give the attackers a false benefit and lure them away from legitimate targets while the defender observes the attacker's actions and gathers information on their behavior. Example deception techniques within the honey-object family include the following:

- **Honeypots:** Decoy systems that are designed to appear identical to a real system [3].
- **Honeyservices:** Fake services that appear to be running on a system (e.g. fake SSH server, webserver, file server, etc.)
- **Honeytokens:** Fake access credentials that appear to be legitimate but should trigger an alert if used, since no real user should be using them, thus their use would indicate or imply malicious activity [4].
- **Honeyfiles:** Bait files that appear to be valuable and intended for an attacker to open them, setting off an alarm [5].

- **Honeyusers:** Fake users on a system that are intended to be used or interacted with by an attacker, triggering an alert upon such an event.

2.1.2 Use Cases and Applications

Deception techniques do not solve a particular problem; they are highly versatile tools that have many broad applications to security and come in many forms. Deception techniques all share a common feature: they all have no production value and, thus, should see no activity in a normal setting [6]. Any activity on them likely implies malicious intent. As such, deception techniques can be used to achieve a variety of goals, including the following:

- **Intelligence Collection:** Deception techniques, such as honeypots, allow organizations to gather valuable information about attackers including their techniques, tactics, and procedures. This information can be used to understand the attacker's motives and methods, and develop more effective defense strategies to protect the organizations from future attacks. For example, an organization can deploy a honeypot which, once the attacker infects, can be used to collect about the attacker's actions. This data can then be used to create a profile of the attacker's tactics and behavior, allowing the organization to switch to a more offensive strategy [7].
- **Enhancing Intrusion Detection Systems:** Deception techniques are used to enhance the capabilities of intrusion detection systems (IDS) by focusing on the attacker's perception of the system and anticipate an attack before it takes place [8].
- **Wasting Attacker Resources:** Deception techniques, such as honeyservices, are used to waste an attacker's time and resources by luring them away from legitimate targets, giving defenders additional time to locate and respond to the attack. When responding to an attack, every second of additional time is precious to the incident response team; every second the attacker is wasting on a decoy is a second that it is not attacking the real system.
- **Deterrence:** If an attacker knows that a network or a system is using deception techniques to protect its resources, they may become more wary about attacking it since they are not certain if the information they are gathering is legitimate or not. If done strategically, this may deter some opportunistic attackers from attacking the system, thus reducing the number of attacks that the defender must respond to [9].

2.1.3 Limitations

Using deception techniques is not without its drawbacks; deception techniques are not a silver bullet and are not guaranteed to be effective in all situations.

Attackers Must Interact with Deceptive Objects

For all deception techniques within the honey-object family, the attacker must interact with the deceptive object in order for them to be useful and for the defender to gather intelligence about the attacker. Attackers and malware are become increasingly sophisticated and are able to detect and avoid honeypots [10].

Attackers May Exploit Deceptive Objects

Naturally, this risk does not apply to all deception techniques, as simpler deception techniques (such as honeytokens) are not considered exploitable. However, more complex methods, such as honeypots, carry more risk. An attacker is meant to interact with a honeypot in a way that a defender can learn more about their objectives and behaviors. However, with honeypots, there is a risk that the attacker may be able to exploit the system and use it as a vector to other systems in the network. With correct separation and DMZ's, this risk can be mitigated [10], but the benefits of the tool must be weighed against the risks.

2.1.4 Is Deception Relying on Security Through Obscurity?

Security through obscurity refers to the practice of protecting information by making it difficult to understand or access, such as using different daemon ports (like SSH on port 2222 instead of 22), hiding software versions, or obfuscating code by making it difficult to read and, thus, hack. The idea is that if an attacker cannot easily figure out how something works, they will be less likely to exploit it. However, this is generally considered a bad practice particularly if used as the sole security system, as it is not a substitute for proper protection measures, and can give a false sense of security.

Deception techniques such as honey-objects do share some similarities with security through obscurity, as both involve hiding or misleading information. However, there are several key differences between the two. While security through obscurity typically aims to conceal the true nature of security mechanisms, deception *invites* interaction with false targets to collect intelligence and learn about attackers' behavior. Additionally, deception does not rely on hiding the real security mechanisms; it adds a layer of complexity without necessarily concealing the true nature of the system. Furthermore, deception involves active engagement with attacker, monitoring their activities, and learning from them whereas security through obscurity simply aims to hide information. Finally, deception is not meant to replace standard security measures, rather it serves as a valuable supplement and complement to them [10].

2.1.5 How Effective is Deception?

Although promising, the efficacy and effectiveness of deception in real environment is unclear. Evaluating the efficacy of deception is a notoriously difficult task due to the scale of the scenario space and other factors that are difficult to measure. Specifically, deception efficacy can vary greatly due to a number of factors, including (but not limited to) the network topology, the type and combinations of deception techniques used, the attacker threat model, the defender capabilities and resources, and the network vulnerabilities. Additionally, some features such as the *believability*, *indistinguishability* and *secrecy* of deception techniques are difficult to measure or formalize [11].

Currently, no deception evaluation platform effectively and meaningfully evaluates deception techniques because they either are not flexible enough to evaluate a wide variety of scenarios, or rely on human subjects as the adversary, making experiments expensive and time-consuming in addition to being difficult to reproduce [12]. We elaborate further in Chapter 3.

Chapter 3

Related Work

Research in deception is still in its infancy, and there is still much to be explored [13]. One of the main challenges in deception is the lack of a comprehensive framework for evaluating the efficacy of deception techniques. As a result, many researchers have already begun exploring different approaches to evaluating deception effectiveness. In this section, we discuss the current state of cyber deception research, and the limitations of existing deception evaluation methods and platforms.

3.1 The Human Approach

One approach to evaluating deception is to use experienced human subjects as the attacker against a system that uses deception techniques to protect its resources. In the Moonraker paper, Shade *et al.* [12] conducted one of the first series of rigorous, controlled experiments using human subjects to examine the effectiveness of deception for cyber defenses. They hypothesized that adding deceptive actions would impede the progress of an attacker and create a more time-consuming and frustrating experience for them. In their experiments, they used human subjects with varying levels of experience in cyber security who were all told to execute six tasks that the researchers would observe.

While this approach was indeed effective at evaluating the effectiveness of deception techniques, it was limited by the population from which the participants were drawn and the skill of each individual. Additionally, using humans in the evaluation process is expensive and time-consuming, and is difficult to reproduce. Furthermore, it is difficult to evaluate specialized scenarios since it is challenging to find individuals with the necessary technical skill and proficiency to perform attacks in such scenarios. Despite this, [12] was one of the first papers to rigorously explore the human approach to deception evaluation.

Other researchers have also explored the human approach to deception evaluation, including Ben Salem and Stolfo [14], Voris *et al.* [15], Aggarwal *et al.* [16], Aggarwal *et al.* [17], and Ferguson-Walter *et al.* [18]. However, all of their papers suffer from similar limitations as [12], with [14] and [15] using computer science students as the test subjects, who typically have less experience than cybersecurity professionals.

3.2 Game Theory and Model Approaches

Another approach to evaluating deception techniques is through **deception modeling**, which refers to the set of rules and strategies that govern the defender's integrating of deception within the architecture of a target system [11]. While searching for novel evaluation methods, we found different approaches to modeling deception. In the study by Carroll and Grosu [19], the researchers proposed a method that utilized game theory models to describe various interaction sequences between the attacker and the target system, and to apply deception whilst optimizing the defender's gain.

Additionally, the study by Cohen and Koike [20], the researchers proposed a different approach which leveraged the use of attack graphs to provide a concise representation of known attack scenarios and the attack vectors that can be used by malicious parties. The researchers wove deception techniques into the graph to lure attackers towards fake targets and evaluate their effectiveness.

Finally, the study by Anwar and Kamhoua [21] introduced a novel approach centered around attack graphs intertwined with game theory principles, seamlessly embedding deception mechanisms within the attack graph structure. This integration not only provides a comprehensive representation of potential attack pathways but also strategically misdirects adversaries by guiding them away from genuine assets and towards deceptive nodes.

Modeling approaches to deception evaluation present a set of challenges that researchers and practitioners must grapple with. Naturally, models are limited by the assumptions made by the researchers, and the accuracy of the model is dependent on the accuracy of those assumptions. Accurately representing the intricacies of real-world deceptive tactics within a model can be quite complex, often leading to an explosion in the size and complexity of the model in addition to oversimplifications that may not effectively capture a real attacker's behavior. Furthermore, the dynamic nature of cyber threats and the ever-changing nature of the cyber landscape means that static models can quickly become outdated, requiring constant updates to remain relevant.

3.3 Deception Evaluation Platforms

While exploring the literature around deception evaluation, we did not find platforms that specifically *emulate* attackers and defenders for the sake of evaluating deception techniques. However, we did find several platforms that allow users to create and manage emulated agents and environments [22], [23], in addition to a platform that evaluates deception techniques with a network attack *simulation* [24]. **Simulation** replicates the behavior of a system using abstract models, while **emulation** mimics the exact functionalities of one system using another system.

CyGIL, the platform proposed by Li, Fayad, and Taylor [23], is an experimental, emulated testbed designed for reinforcement learning (RL) in the domain of cyber operations. CyGIL employs a stateless environment architecture and integrates the MITRE ATT&CK framework to create a high-fidelity training environment. The platform allows a user to add their own functionality, such as deception capabilities and evaluation methods, but it is not purpose-built for deception evaluation.

Similarly, CybORG, the platform proposed by Baillie *et al.* [22], is a research gym tailored for Autonomous Cyber Operations (ACO) focusing on the autonomous decision-making and action in cyber defense, encompassing both the attacker and defender dynamics. CybORG is designed to support the application of RL and offers both simulation and emulation modes. While it is also flexible enough to allow a user to add deception capabilities that may enable the user to evaluate defensive method, it is not purpose-built for deception evaluation either.

Finally, the platform proposed by Reti *et al.* [24] is a deception evaluation platform that uses a network attack simulation to quantitatively measure the effectiveness of deception techniques, providing insights on the optimal honeypot deployment and the ideal intervals for mutating network addresses to maximize the disruption of potential attacks. While this platform is indeed purpose-built and does offer extensibility for the user to add their own deception techniques, it is limited by the fact that simulation-based approaches may not capture the full complexity of real-world attacks as accurately as emulation-based approaches.

PERRY aims to address the limitations of existing deception evaluation platforms by providing a flexible, extensible, and easy-to-use platform that allows users to evaluate deception techniques in a realistic and automatically instantiated environment. We describe the system design in Chapter 4 and the implementation Chapter 5.

Chapter 4

System Design

In this section, we will detail the design of our system by outlining the desired properties, exploring the components involved, and navigating the overall layout and interaction of the system from a high level, which we will then expand upon by describing the implementation in Chapter 5.

4.1 PERRY's Properties

We wanted PERRY to be able to help security researchers and practitioners quickly test out and evaluate the efficacy of their deception techniques against several types of attacks. In order to achieve this, we came up with the following properties that PERRY should have:

- **Realistic:** PERRY should be able to create an environment that emulates a real network, in addition to being able to emulate a realistic attacker.
- **Extensible + Flexible:** PERRY should be able to support a variety of different attacks, exploits, vulnerabilities, deception techniques, network topologies, and goals, and make it easy to extend the system by adding new features and properties to any of the components. Additionally PERRY should be flexible enough to allow the user to easily configure the system to suit their needs and applications.
- **Automated Instantiation:** PERRY should be able to quickly and automatically instantiate the environment and scenario with minimal user input, allowing us to quickly evaluate many different types of defenders without having to manually execute each scenario.

4.2 System Abstraction

In this section, we explore the high-level design of PERRY, describing each primary component, their purpose, and their interactions with each other.

As is common in cybersecurity, we designed our system similar to game theoretic models of security games, in which two players, typically an attacker and a defender, interact strategically in order to achieve their goals. While we don't formally model our system as a game, we do use similar concepts and abstractions as they relate specifically to network security [25].

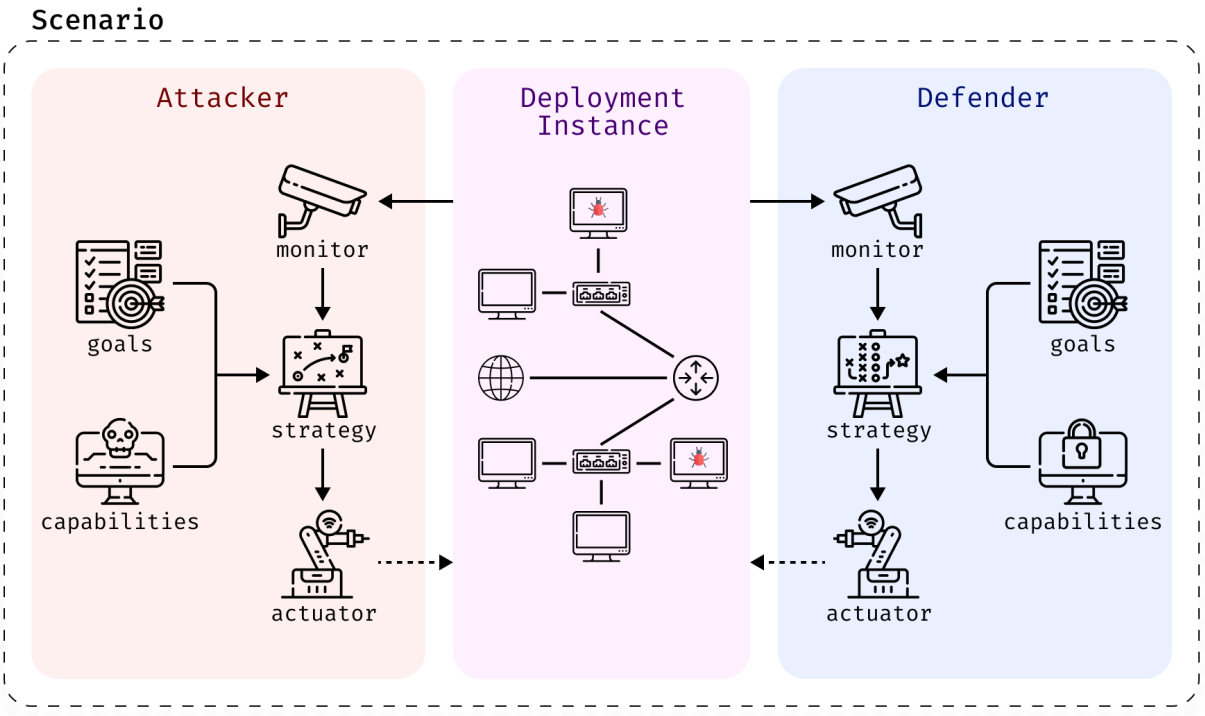

Figure 4.1: High-level system abstraction of PERRY, depicting the main components in a scenario, their subcomponents, and their interactions with each other. The solid arrows represent the flow of information, and the dashed arrows represent the execution of actions.

As shown in Figure 4.1, an instance of a PERRY scenario contains the following components:

- **Deployment Instance:** The deployment instance is the "battlefield" upon which the attacker and defender will be fighting.
- **Attacker:** The attacker is the adversary that is trying to gain access to the network.
- **Defender:** The defender is the entity that is trying to protect the network from the attacker.

4.2.1 Deployment Instance Abstraction

The deployment instance is defined by a **Network Topology**, which specifies the arrangement of hosts on a network, a **Network Configuration**, which specifies the network security groups and rules, and a set of **Vulnerabilities**, which specifies the vulnerabilities that will be deployed onto certain hosts in the network (represented by the red bugs in Figure 4.1). The deployment instance is the environment that contains all the valuable resources that the attacker is attempting to gain access to, and the defender is trying to protect by deploying deception techniques on.

4.2.2 Attacker and Defender Abstraction

The attacker and the defender of a scenario are abstracted identically and are represented by the following components:

- **Monitor:** The monitor is the component that monitors the deployment instance network and reports its observations.
- **Goals:** The goals are the set of objectives that the attacker/defender is trying to achieve.
- **Capabilities:** The capabilities are the set of actions that the attacker/defender can perform or apply on the network.
- **Strategy:** The strategy is the process that determines what next action should be taken. Given the current state of the network (received as input from the monitor), the attacker/defender strategy will determine which capabilities to use in order to achieve its goals and outputs the set of actions to be performed.
- **Actuator:** The actuator is the component that executes the given set of actions on the deployment instance.

While the attacker and defender have identical abstractions, their implementations are quite different.

Evaluation Criteria and Metrics of Success

To properly understand the effectiveness of deception techniques, our attacker and defender's goals incorporate a notion of 'score' which they each keep track of and try to optimize. Both the attacker and the defender have a 'goalkeeper' (not explicit in Figure 4.1 but implied by the **goals** component) that keeps track of the score and determines whether they have achieved their goals. The way the score is calculated and the way the goals are achieved are not predetermined and can be defined by the user. For example, the attacker's goal could be to gain access to a certain host in the network, and the defender's goal could be to prevent the attacker from achieving its goal.

Chapter 5

System Implementation

In this section, we build upon the system abstraction described in Chapter 4 and describe the implementation details of PERRY. For each component, we give a brief introduction of its purpose and significance. We then detail the technical foundations and the tools used to implement each component, after which we describe the integration mechanics and how the various tools synergize to realize the component's implementation. Finally, we take a deep dive into the implementation details of each component. Note that for the sake of brevity, we will discuss the most important aspects of the implementation, and will omit the less significant details. Additionally, it may be useful for the reader to refer to the system abstraction in Figure 4.1 while reading this section.

5.1 Scenario Specification

In PERRY, a **scenario** represents a specific configuration encompassing three primary entities: an attacker, a defender, and a deployment instance. In order to facilitate the creation and management of these scenarios, we utilize a **scenario specification** file, which serves as a comprehensive blueprint, detailing every facet of the scenario. It specifies the exact deployment instance to be employed and the exact profiles of both the attacker and the defender. These profiles are crucial as they define the capabilities, strategies, and goals each entity possesses, such as the types of attacks an attacker can launch or the defensive measures a defender can deploy. By providing a clear blueprint of the environment and the behaviors of the involved entities, the scenario specification file ensures that each emulation conducted is performed in a controlled, replicable, and well-defined manner.

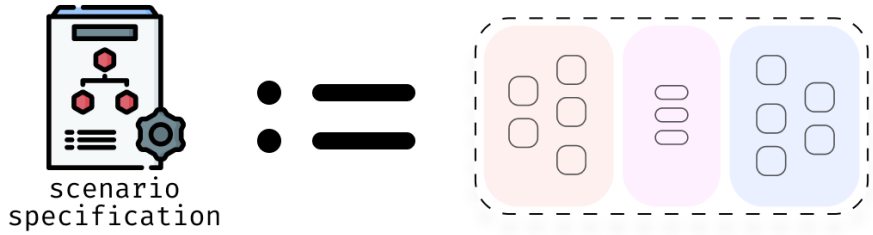

Figure 5.1: Scenario specification files define everything about the scenario (attacker, deployment instance, and defender).

5.2 Deployment Instance

The deployment instance serves as the foundational environment upon which our scenarios are built and executed. It represents a meticulously crafted digital landscape, encompassing network topologies, host configurations, and vulnerabilities. This landscape is the environment in which the attacker and defender will interact, and as such, it is crucial that it is realistic and representative of a real-world network. In order to achieve this, we need a tool to define the landscape, and a tool to instantiate and manage it.

5.2.1 Technological Foundations

For the deployment instance, we utilize a combination of Terraform and Openstack to define and provision the network resources, and Ansible to configure the hosts and install vulnerabilities.

Configuration File

To ensure flexible and rapid deployment, network resources and topologies are created as a human-readable configuration file, which can easily be altered to adapt to different deployments and requirements. The configuration file defines the network topology, resources, configuration, policies, and security groups.

Vulnerabilities

The vulnerabilities are defined simply as a mapping from a host to a vulnerability. The host is specified by its IP address, and the vulnerability by its name.

Terraform

Terraform is an open-source tool created by HashiCorp that allows us to define and provision infrastructure as code [26]. It interprets the configuration file to automatically provision network resources on the tool that will instantiate and manage them [27]. We chose Terraform because it facilitates rapid and flexible deployment, allowing us to easily define various network topologies and configurations.

Openstack

OPENSTACK is our tool of choice for the instantiation and management of the network landscape and environment. It is an open-source, cloud-computing platform that allows us to create and manage virtual machines, networks, and other cloud resources [28]. We chose OPENSTACK because it is open-source, which allows us to easily deploy it on our own hardware and because it is frequently used in research and in industry. OPENSTACK also has both a command-line and web interface, allowing us to easily interact with it using Python scripts and from a web browser respectively. It also has a great Python SDK and ample documentation, which we use to create and manage the network resources that we defined in our configuration file. Futhermore, OPENSTACK satisfies our needs for realism and flexibility in addition to immediately and automatically instantiating the network resources, allowing us to quickly deploy and update hosts and network components.

Ansible

ANSIBLE is a tool that offers powerful task and deployment automation [29]. Tasks are scripted in ANSIBLE playbooks, which are then executed on remote hosts. We chose ANSIBLE for its capability to quickly and efficiently run numerous commands on remote hosts, thus meeting our requirements for flexibility, extensibility, and automation.

5.2.2 Integration Mechanics

All the tools are synergistically integrated to implement the deployment instance. The configuration file is interpreted by TERRAFORM to provision the network resources on the OPENSTACK server. The OPENSTACK Python SDK is then used to retrieve the IP addresses of the hosts, which are then used to connect to and setup each of the hosts and install the vulnerabilities using ANSIBLE scripts.

5.2.3 Implementation Details

The core of the deployment instance is the `DeploymentInstance` class, which serves as the blueprint for each instance. The class integrates the OPENSTACK connection details, the configuration file, the vulnerability map, and the ANSIBLE scripts, culminating to form what we define as the **deployment instance profile**. For each deployment instance profile we want to use, we create a new Python class based on the `DeploymentInstance` parent class, inheriting all of its functions and variables.

First, the configuration file is interpreted by TERRAFORM to provision the network resources on the OPENSTACK server. Following this, the OPENSTACK Python SDK is used to retrieve the IP addresses of the hosts. These IP addresses are crucial as they are pivotal for establishing connections, initiating the installation of base packages, orchestrating the attacker's goal implementation, and embedding vulnerabilities on the hosts via ANSIBLE scripts. Finally, a convenient `setup` function is provided to streamline the setup of the deployment instance.

The specific deployment instance used is determined by the scenario specification file. This modularity enables easy interchangeability of network topologies, configurations, and vulnerabilities, facilitating the rapid evaluation of deception techniques across a broad range of network environments.

5.3 Attacker

The attacker in Perry is an adversary that is designed to emulate realistic cyber threats. The implementation is geared towards flexibility and extensibility, allowing us to easily add new capabilities and strategies to the attacker. Additionally, we emphasize realism in our implementation, as we want the attacker to behave as closely as possible to a real human attacker. In order to do so, we need a tool that will manage the attacker's components, including its strategies, capabilities, and goals. Additionally, we need a system that will realistically emulate the attacker's behavior.

5.3.1 Technological Foundations

In order to implement the attacker, we use Caldera, which handles the management of the attacker's components in addition to providing us with a convenient interface to interact with them. We also implement the attacker's goals as flags on several hosts in the deployment instance.

Caldera

Caldera is an open-source cybersecurity platform that specializes in automating adversary emulation [30]. The attacker in Perry is rooted in Caldera, which has been extended with a custom plugin to incorporate unique abilities and planners for our attacker. Caldera's management of the attacker's monitor and actuator coupled with its convenient interface and SDK to interact with them allows us to easily create a realistic attacker using our own goals, capabilities, and strategies. We chose Caldera because it is open-source, and emulates a real attacker as closely as possible, which is crucial for the realism of our platform. Additionally, Caldera is highly flexible and extensible, allowing us to easily add new features and capabilities to it. Finally, Caldera automatically instantiates the attacker, allowing us to quickly deploy and update the attacker as needed.

Flags

For the sake of simplicity, we implemented the attacker's goals as flags. A **flag** is a text string that is discreetly hidden on intentionally vulnerable targets that the attacker is trying to find, most commonly used in Capture the Flag (CTF) exercises. We plant flags on hosts within the deployment instance, and the attacker captures a flag simply by reading the flag file. Since we do not strictly define the attacker's goal as anything specific, we are not limited to using flags, and can redefine the objective to something else as needed, such as the attacker trying to gain access to a specific user, or to obtain database root credentials, and so on.

5.3.2 Integration Mechanics

All the tools are integrated to implement the attacker. CALDERA is used to automatically and realistically emulate the attacker using our strategies, which interact with the attacker's monitor and actuator using our capabilities. The attacker's goals are implemented as flags on hosts within the deployment instance, which it will attempt to capture as per its strategy.

5.3.3 Implementation Details

The CALDERA framework consists of two primary components: the **core system** consisting of an asynchronous command-and-control (C2) server with a REST API and web interface, and the **plugins** that define the attacker's capabilities and strategies [31]. We extend CALDERA by creating our own plugin with custom abilities and planners for our attacker. The attacker's capabilities are implemented as CALDERA abilities with custom parsers, and the attacker's strategy is implemented as a CALDERA planner. For each desired attacker profile, a new CALDERA planner Python script is crafted, outlining the attacker's goals and strategies. Additionally, a configuration file is created, listing all of the attacker's available capabilities. Finally, for each attacker, we create a Python class that links the attacker's planner script to the capability configuration file and points to the CALDERA server, culminating to form what we define as the **attacker profile**. This class is then used to instantiate the attacker. The flexibility of this setup enables swift and easy modifications by simply changing the attacker profile in the scenario specification file, thus allowing us to quickly test out different attacker profiles and assessing their impact on the defender's strategies.

The infection process for our attacker is as follows. Once the attacker is installed on a host and the operation is started, the attacker will scan the network for vulnerable hosts and select a host at random for its attack. The host is then scanned, leading to a selection of open ports, which the attacker exploits based on the active service on the port and the capabilities available to the attacker. Upon gaining access to a host, the attacker attempts to read the host's flag before laterally moving to infect another host. The infection mechanism involves the installation of a CALDERA agent on the host, essentially creating another instance of the attacker within the same CALDERA operation. This requires the attacker to be able to communicate with the CALDERA server, which we allow in our network configuration.

5.4 Defender

The defender in PERRY serves as a crucial component, designed to maintain knowledge of the network and respond to potential threats. Similar to the attacker, we emphasize automation, realism, and flexibility in our implementation such that we can easily add new capabilities and strategies to the defender. In order to do so, we need a tool that will collect telemetry from the deployment instance, a tool that will manage the defender's knowledge base, and a tool that will manage the defender's strategies and capabilities. Finally, we need a tool that will allow us to apply the defender's actions onto the deployment instance.

5.4.1 Technological Foundations

In order to implement the defender, we use SYSFLOW and ELASTICSEARCH to collect telemetry from the deployment instance and to maintain the defender's knowledge base. We also use ANSIBLE and the OPENSTACK Python SDK to apply the defender's actions onto the deployment instance. Finally, we create a Python module to manage the defender's strategies and capabilities.

SysFlow

SYSFLOW is an open-source telemetry framework for monitoring cloud workloads and facilitating the creation of performance and security analytics [32]. One of its key features is that it can collect system call and event information from hosts and export them in a specific format to a desired endpoint. We chose SYSFLOW because it is open-source, which allows us to easily deploy it on our own hardware, and because it is frequently used in research [33]. Finally, SYSFLOW gives us the ability to collect telemetry from the deployment instance, which is essential for monitoring the network.

Elasticsearch

ELASTICSEARCH is an open-source, distributed search engine that allows us to store, search, and analyze large amounts of data quickly and in near real-time [34]. We chose ELASTICSEARCH because it is open-source, which allows us to easily deploy it on our own hardware and because it is frequently used in research and in industry. Additionally, it is highly flexible and extensible, featuring a huge library of plugins and integrations. Finally, ELASTICSEARCH automatically indexes the telemetry events, allowing us to quickly search and analyze them, which is why we use it to maintain the defender's knowledge base.

Ansible

ANSIBLE is a tool that offers powerful task and deployment automation. We have already discussed ANSIBLE in Section 5.2.1. For the defender, we use ANSIBLE playbooks to quickly and easily apply the defender's actions onto the deployment instance.

Openstack

OPENSTACK is an open-source, cloud-computing platform that allows us to create and manage virtual machines, networks, and other cloud resources. We have already discussed OPENSTACK in Section 5.2.1. For the defender, OPENSTACK serves as the interface to the deployment instance, allowing us to easily retrieve information about the network and hosts, and also to apply the defender's actions onto the deployment instance.

Defender Python Class

The core of the defender is rooted in the PERRY `Defender` class, which serves as the blueprint for each defender. This class manages the strategy, capabilities, and goals of the defender.

5.4.2 Integration Mechanics

Sysflow and Elasticsearch operate in synergy, with Sysflow collecting telemetry data and exporting it to Elasticsearch for storage and analysis. The defender, implemented in Python, leverages this data to make strategic, informed decisions and execute actions. The defender's strategy within the Python class dictates the appropriate response to telemetry events based on the defender's available capabilities. Finally, the defender's actuators leverage Ansible and the Openstack Python SDK to apply the actions onto the deployment instance.

5.4.3 Implementation Details

The core of the defender's implementation is encapsulated within a series of meticulously designed Python classes, each tailored to serve a specific purpose within the defender's overall architecture.

Defender Profile Class

Central to the defender's operation is the `Defender` class which serves as a blueprint for each defender. The class integrates connections to the Elasticsearch server, the Openstack server, and the Ansible scripts, culminating to form what we define as the **defender profile**. For each desired defender profile, a new Python class is crafted based off the `Defender` parent class, inheriting all of its functions and variables. The class describes the defender's overarching strategy in addition to housing the telemetry analysis processor, which is pivotal in processing telemetry events that subsequently dictate the defender's actions. Within this class, the defender's goals are also defined, which are used to determine the defender's overall success.

Capability Modules

The defender's capabilities, which are the actions and responses that the defender can take, are implemented as distinct Python modules. Each module defines the parameters required by the actuator to execute the associated action. This modular approach ensures that adding new capabilities or modifying existing ones is a simple and streamlined process.

Actuator Modules

Complementing the capability modules are the actuator modules. These are the refined, specialized workers of the defender, translating the strategies and capabilities into tangible actions performed on the deployment instance. Each capability has an associated actuator module responsible for executing the action it defines. The implementation of the actuator is flexible enough to allow us to use a variety of different methods to apply the action, such as using the Openstack Python SDK in conjunction with Ansible scripts to download a git repository and execute a Python script on a host. For instance, the actuator associated with the `DeployHoneyservice` capability would use the Openstack Python SDK to interface with the Openstack server to connect to a host and execute an Ansible script to download a git repository containing the honeyservice program, which it then executes on the host. The design of these actuators is inherently flexible and extensible, allowing for a diverse range of methods to apply actions.

Telemetry Analysis Processor

The telemetry processor seamlessly integrates with the ELASTICSEARCH and OPENSTACK servers, which is facilitated through dedicated classes that handle communication and data retrieval. The telemetry processor queries the ELASTICSEARCH server for telemetry events that are relevant to the defender and parses them into a format that is easily digestible by the defender. The telemetry processor also queries the OPENSTACK server for information about the deployment instance, such as the IP addresses of the hosts, which is used to determine which hosts to apply actions on. The telemetry processor is the core of the defender, as it is responsible for processing telemetry events and determining the defender's next action. The telemetry processor is also responsible for determining the defender's success, which is based on whether or not the defender has achieved its goals.

In essence, the implementation of the defender is a harmonious blend of extensible and flexible Python classes, each serving a distinct and specialized role, yet collectively ensuring that the defender operates efficiently and robustly. This implementation emphasizes the importance of modularity and extensibility, as it allows us to easily add new capabilities and strategies to the defender, and to quickly test out several different defender profiles.

5.5 Emulator

The emulator serves as an integral component, orchestrating the interactions between the attacker and defender in the deployment instance. Designed as a python program, the emulator offers a command-line interface facilitating the loading of a scenario that will be loaded and executed. As such, the emulator requires a parser to interpret a configuration file, a controller to manage the emulation process, and an orchestrator to instantiate the scenario.

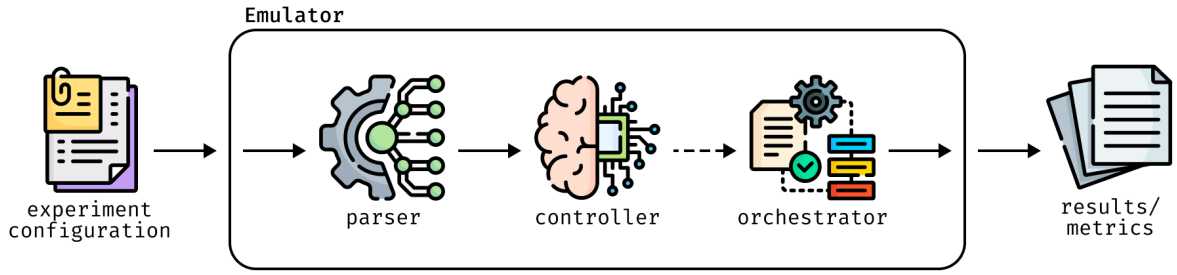

Figure 5.2: Emulator Process Diagram. Solid arrows represent the flow of information, and dashed arrows represent invocation of entities.

5.5.1 Technological Foundations

We first introduce an **experiment configuration file** that specifies a list of scenarios to emulate. We then need a **parser** to interpret the experiment configuration file, a **controller** to manage the emulation of a scenario, and an **orchestrator** to instantiate the scenario.

Experiment Configuration File

This file specifies a list of experiments to run. Each experiment defines the scenario specification file to use in addition to a number of other critical properties. This allows us to easily run many experiments with different scenarios and parameters without having to manually execute each one. This file is distinct from the scenario specification file, which defines the scenario to emulate, as described in Section 5.1.

Parser

The parser is the component of the emulator that interprets the experiment configuration file and creates a list of all the experiments to run and their details, including the number of trials for each experiment, which scenario specification file to use, the output directory to save the results to, how to handle errors in a trial, and which OPENSTACK, ELASTICSEARCH, and CALDERA servers to use. This allows us to easily run many experiments with different scenarios and parameters without having to manually execute each one.

Controller

The controller is the component of the emulator that manages the emulation of a scenario. It is responsible for loading the scenario specification file, invoking the orchestrator, running the scenario, saving the results, and repeating the process for the specified number of trials for each experiment.

Orchestrator

The orchestrator is the component of the emulator that instantiates the scenario. It is responsible for importing the required classes and setting up the deployment instance, attacker, and defender.

5.5.2 Integration Mechanics

The parser parses the experiment configuration file to create a list of experiments to run, which is then passed to the controller. The controller then invokes the orchestrator with the experiment information, which then uses that information to instantiate the scenario. The orchestrator then returns the scenario to the controller, which then runs the scenario. The controller then saves the results and repeats the process for the specified number of trials for each experiment.

5.5.3 Implementation Details

The emulator provides a command-line interface that can be used to load an experiment configuration file, load a particular scenario specification file as an experiment, begin emulation, and view all results. The implementation of the emulator centers around flexibility, extensibility, and automation; it can easily be extended to add new commands and features, and emphasizes automation by attempting to minimize as much manual work as possible when running experiments and emulating scenarios.

The controller is the core of the emulator. Once a list of experiments is received from the parser, it will iterate through each experiment and invoke the orchestrator with the details of the experiment. The orchestrator will import the specified profiles for the deployment instance, attacker, and defender, and instantiate each of them, thus defining a scenario for the controller. The controller will then run the scenario, which will involve the attacker attempting to achieve its goals and the defender attempting to prevent the attacker from achieving its goals. Once the scenario is complete, the controller will save the results and repeat the process for the specified number of trials for each experiment.

Additionally, since the process of setting up the deployment instance is a lengthy one, the controller can be used to save some time between trials. We give the controller the ability to save a snapshot image of each host after it has been set up the first time. Therefore, between trials, the controller can restore all the hosts to a clean state by rebuilding them from their saved images, then performing minimal setup such as installing the attacker agent and planting fresh flags. This allows to easily run hundreds of trials of an experiment without having to wait for the network to be deployed and the hosts to be set up each time.

Chapter 6

Experiments and Results

6.1 Experimental Setup

In this section, we describe the experimental setup that we used to evaluate the efficacy of several deception techniques against several types of attacks. Additionally, we describe the network topology that we used, the vulnerabilities that we deployed, and the attacker's strategy. We also describe the defender's strategy, which we will vary between experiments.

To come up with the experiments, we used deception techniques that we found in literature and adapted them into defender strategies that we thought would be interesting to test out.

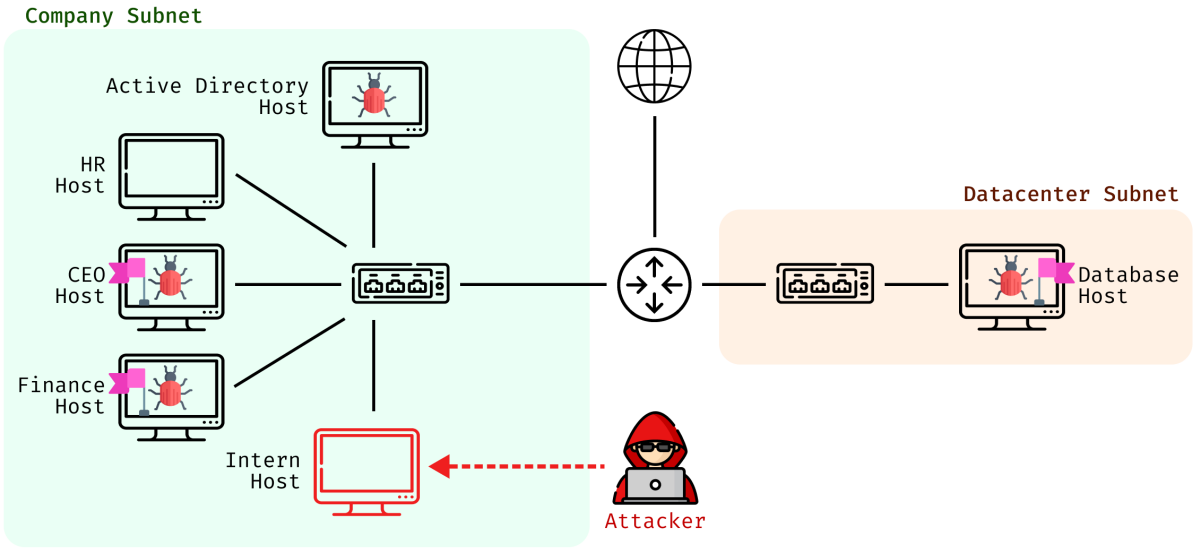

Figure 6.1: Experimental setup of the network topology, showing the hosts, subnets (colored square backgrounds), flags (magenta flags on hosts), vulnerabilities (red bugs on hosts), and the attacker's initial position (red-colored host).

6.1.1 Hardware and Software Information

We ran our experiments on a single Ubuntu machine with 128 GB of RAM, Intel(R) Xeon(R) W-2295 processor with 18 cores and a base frequency of 3.00 GHz. Additionally we used TERRAFORM `v1.3.7`, a forked version of CALDERA `v4.1.0` (due to some bugs in the original CALDERA project), OPENSTACK `Zed` (released 05 October 2022), ANSIBLE `core v2.15.0`, ELASTICSEARCH `v8.6.2`, and SYSFLOW `v0.5.1`.

6.1.2 Deployment Instance Setup

For our setup, we created a network topology as shown in Figure 6.1 that contains two subnets: the `company` subnet and the `datacenter` subnet. On the company subnet, we added five hosts: `ceo`, `hr`, `finance`, `activedir`, and `intern`. On the data center subnet, we added a single host: `database`. The two subnets are connected to each other via a gateway router, which connects to the outside world.

For the sake of simplicity, we allowed full communication between the two subnets; we setup security rules such that all hosts on the `company` subnet can communicate with all hosts on the same subnet and the `datacenter` subnet, and vice versa via, TCP on ports `1-65535`. These rules also apply to any new hosts that may join the network. Additionally, we allowed the attacker to communicate with our CALDERA server.

Furthermore, we planted three flags on each of `ceo`, `finance`, and `database`, as depicted by the magenta flags on the hosts in Figure 6.1, which the attacker will attempt to capture. These flags are the implementation of the attacker's goals, as described in Section 5.3. Finally, we installed several vulnerabilities, detailed in Table 6.1 and depicted by the red bugs on the hosts in Figure 6.1, which the attacker will attempt to exploit in order to gain access to the host.

Table 6.1: Vulnerability Configuration for Our Deployment Instance

Host	Vulnerabilities
`activedir`	`netcat` Bind Shell Listener on Port `4444`
`ceo`	SSH Login with Password
`finance`	`netcat` Bind Shell Listener on Port `4444` `vsftpd` Backdoor [35]
`hr`	None
`intern`	None (Initial Attacker Access)
`database`	`netcat` Bind Shell Listener on Port `4444`

6.1.3 Attacker Setup

We kept the attacker constant for all of our experiments in addition to keeping the scoring value of each flag identical. Furthermore, the attacker has no prior knowledge of where the

flags are, and will attempt to search for them on every host it infects. Importantly, the attacker will not change its behavior based on the defender's strategy nor will it value one flag or host over another. We also assume that the attacker cannot distinguish between real and fake hosts, or between real and fake services, thus the attacker *will* interact any with honey-objects the defender deploys if it encounters them. We discuss the ramifications of this in Section 7.1.2 and leave the task of creating an improved attacker for future work.

The attacker begins on `intern` by scanning both the `company` and `datacenter` subnets to gather all the online hosts on each of them. After that, the attacker randomly scans each online host for open ports, and if it finds any, will attempt to run an exploit based on the type of service running on that port. If the attacker successfully exploits a vulnerability, it will "infect" the host by installing an instance of itself on it (CALDERA attacker agent) effectively gaining full control of the host. If the attacker is unable to exploit any vulnerabilities on the host, it will move on to the next host. The attacker will continue this process until it is able to capture all three flags, or until it is unable to find any more vulnerabilities to exploit. The attacker "captures" a flag by reading the contents of the flag file. For the sake of our experiments, once the file has been read, we consider the flag to be captured even if the attacker does not exfiltrate the flag string.

The attacker's goal is to capture all three flags as quickly as possible. Thus, the attacker's **metrics of success** are the number of flags captured and the execution time (namely the time taken to capture all flags that the attacker is able to capture).

6.1.4 Defender Setup

In our experiments, we wanted to test out several types of defender techniques. To achieve this, we created four different defender profiles, generally defined as follows:

0. **No Defender (Baseline):** This defender does nothing, effectively acting as a baseline for the attacker's performance.
1. **Passive Defender:** This defender has several deception capabilities that it deploys at the start of the experiment. The purpose of this defender is to create more noise that the attacker will be forced to sift through in order to find the flags.
2. **Active Defender:** This defender has the same deception capabilities as the passive defender, but with an added alert system. Specifically, if the attacker interacts with and of the deployed deception capabilities, the defender will be alerted and will restore the alerting host to a clean state, effectively removing the attacker from that particular host. The previous defenders do not receive alerts from the monitor, nor do they have the capability to perform any restorative actions.
3. **Dynamic Defender:** This defender has the same capabilities as the active defender, with the added ability to deploy decoy hosts upon detection of an attacker. The purpose of this defender is to create fake hosts that the attacker will waste time on, thus increasing the time it takes for the attacker to achieve its goals, and giving the defender more time to locate and remove the attacker from the network.

6.2 Evaluation Metrics and Criteria

In order to evaluate the effectiveness of the defender's strategy, we used the following as the **metrics of success** as per the goals of the attacker and the different defender profiles:

- **Number of Flags Captured:** The attacker wants to maximize the number of flags it captures, while the defender wants to minimize how many flags the attacker gains access to.
- **Execution Time:** The attacker wants to minimize the amount of time it takes for it to capture the flags, while the defender wants the attacker to spend as much time as possible.
- **Number of Restores:** *(Active and Dynamic Defenders)* The defender tries to minimize the number of times it restores hosts on the network, since fewer restores means less downtime for the host.
- **Number of Honeypots Deployed:** *(Dynamic Defender)* The defender tries to minimize the number of honeypots it deploys, since fewer honeypots means less resources used.

6.3 Experimental Procedure

For each experiment, we ran hundreds of trials and saved the metrics after each run. These metrics included the number of flags captured and the time that the attacker took to capture the flags. Once all the trials were completed, for each number of flags captured, we calculated the average time it took to capture that number of flags. We then compared the results of each experiment with the previous experiments to determine the effectiveness of the defender's strategy.

6.4 Experiment 0: Baseline (No Defender)

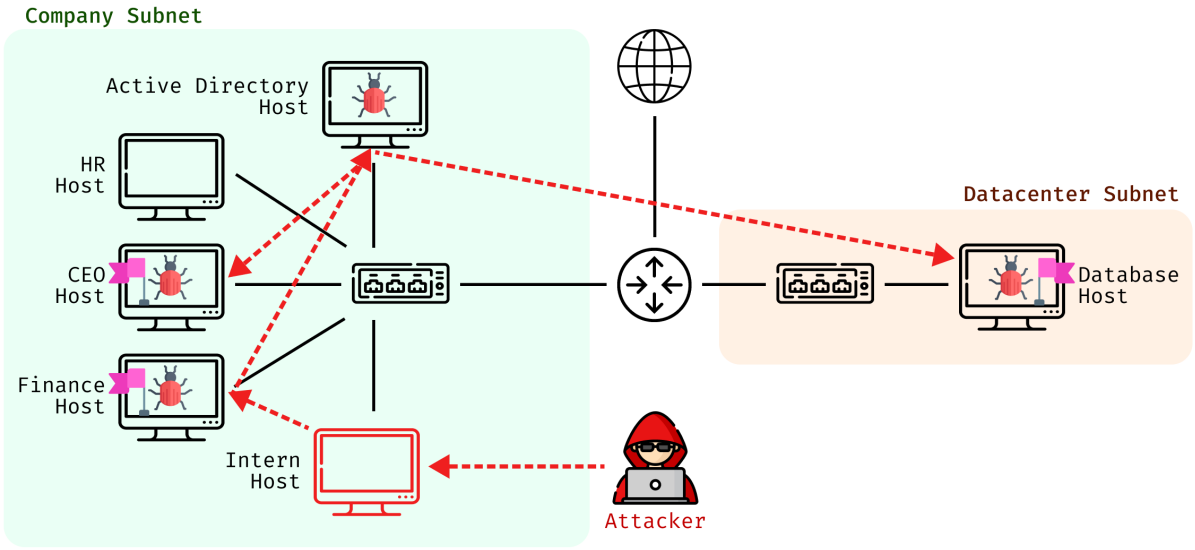

Figure 6.2: Experiment 0 Setup. The red arrows in the network represent a possible path that the attacker can take, not necessarily the one that it *will* take.

In this experiment, we want to establish a baseline for the attacker's performance, which we can use to compare the effectiveness of the defender's strategies in following experiments. This experiment has no defender; the attacker is free to attack the network without any limitations or restrictions other than its own capabilities.

Hypothesis 0 *The attacker will capture all three flags 100% of the time, and will do so in a relatively short amount of time.*

Experiment Outcome

As per the results shown in Tables 6.2 and 6.3 and Figure 6.6, the attacker captures all three flags in 100% of the trials, and did so in an average of 873.3 seconds. This is unsurprising, since the attacker has no defender to battle against, and thus should be able to capture all three flags with relative ease. Hence, our hypothesis was correct.

Experiment 0 Takeaway

The attacker captured all three flags 100% of the time, and did so in an average of 873.3 seconds.

6.5 Experiment 1: Passive Defender

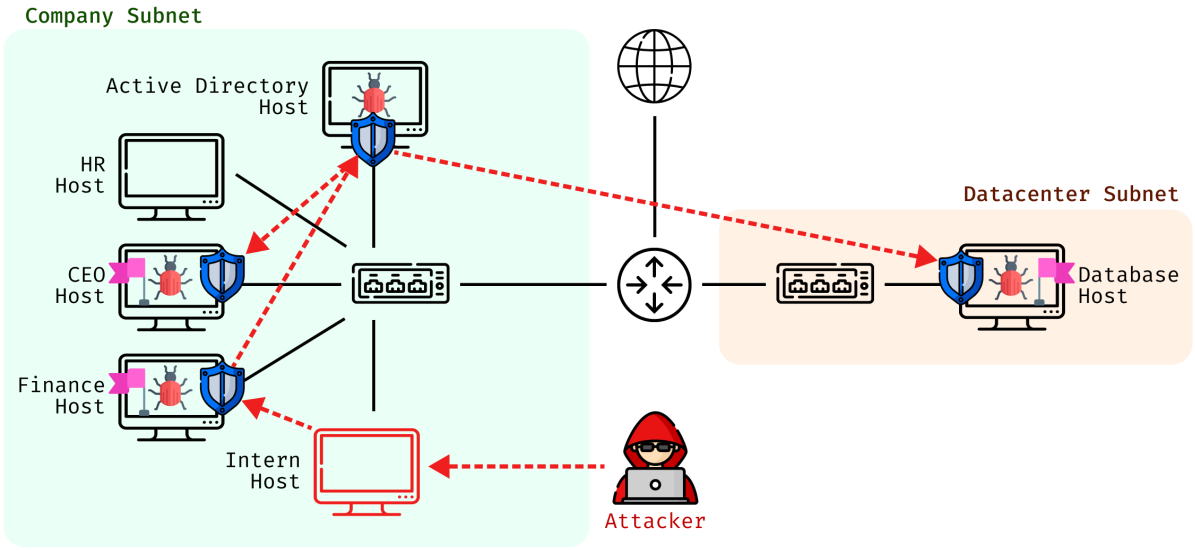

Figure 6.3: Experiment 1 Setup. The blue shields represent a deception capability that the defender has deployed, which, in this case, is a honeyservice.

In this experiment, we introduce a passive defender: a defender that can deploy deception capabilities, which in our case is a honeyservice, and nothing more. This defender does not receive any alerts from the monitor, and thus it cannot respond to any attacker activity. Importantly, we assume that the attacker *will* interact with any honeyservice that it encounters, meaning that the attacker's process should take longer than baseline.

The purpose of this experiment is to determine the effectiveness of deception techniques against the attacker without anything else involved. This allows us to determine the effectiveness of strategies that build upon deception techniques in later experiments.

Hypothesis 1 *The attacker will still capture all three flags 100% of the time, but will take longer on average to do so than the baseline.*

Experiment Outcome

As per the results shown in Tables 6.2 and 6.3 and Figure 6.6, the attacker captured all three flags in 100% of the trials, but did so in an average of 864.59 seconds, which is nearly a 30 second increase from the baseline, equating to a 3.3% increase. This is unsurprising, since the attacker has to sift through the deception capabilities that the defender has deployed, which takes time, however we did hope that the increase would be more significant. Nonetheless, our hypothesis was correct.

Experiment 1 Takeaway

Using deception techniques slowed down the attacker, but did not prevent it from achieving its goals; the attacker still captured all three flags 100% of the time.

6.6 Experiment 2: Active Defender

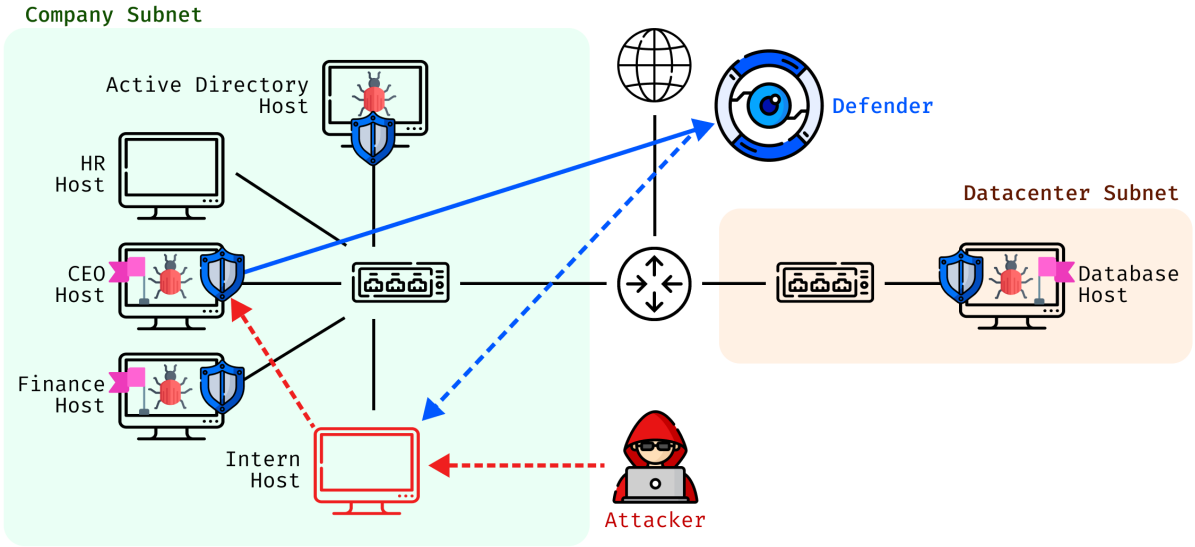

Figure 6.4: Experiment 2 Setup. The solid blue arrow represents an alert being sent to the defender. The dashed blue arrow represents the defender's response by restoring the host interacting with the honeyservice.

In this experiment, we use the same deception techniques as the previous, passive defender, but equipped with telemetry and the added ability to respond to alerts. Specifically, if the attacker interacts with any of the deployed honeyservices, the defender will be alerted of the host interacting with the honeyservice and will restore the host by rebuilding it from a clean image. We assume that a restore removes all traces of an attacker from a host, but we do not assume that the attacker will not be able to infect it again.

Importantly, if the defender restores all machines that attacker infects, the attacker will not be able to regain access to the network. More specifically, in the case that the attacker did not infect any hosts other than the `intern`, and the defender restores the `intern` host, the attacker loses its foothold and will not regain access to the `intern` host. However, if the attacker infects another host before the defender restores it, the attacker will still have access to the network and can continue infecting and reinfecting hosts.

Hypothesis 2 *The attacker will have a significantly harder time capturing all three flags and will take significantly longer on average to do so than against the passive defender.*

Experiment Outcome

Indeed, as per the results shown in Tables 6.2 and 6.3 and Figure 6.6, the attacker captured all three flags only 18.9% of the time, a significant decrease from the passive defender in Experiment 1. Additionally, the attacker took an average of 881.87 seconds to capture all three flags, which is nearly 40 seconds longer than the passive defender, equating to a 5.3% increase, which, similarly to Experiment 1, we expected to be more significant.

Furthermore, the defender prevented the attacker from capturing even a single flag in 42.4% of trials, shutting the attacker down in an average of 224.6 seconds, which is a much better outcome than expected. In the cases that the attacker still captured a flag, we believe that between the time the defender received the alert and the time it performed the restore, the attacker infected another host. This is supported by the fact that the `intern` host is restored more than once per trial on average, as shown in Table 6.4, meaning that, in some cases, the defender found the attacker on the `intern` host multiple times in the same trial.

Since the defender managed to put up a fight, we can conclude that the defender's strategy was effective, and thus our hypothesis was correct.

Experiment 2 Takeaway

Equipping a defender with telemetry and the ability to respond to alerts significantly improved the efficacy of deception techniques, increased the attacker's execution time, and in some cases, prevented the attacker from achieving its goals entirely.

6.7 Experiment 3: Dynamic Defender

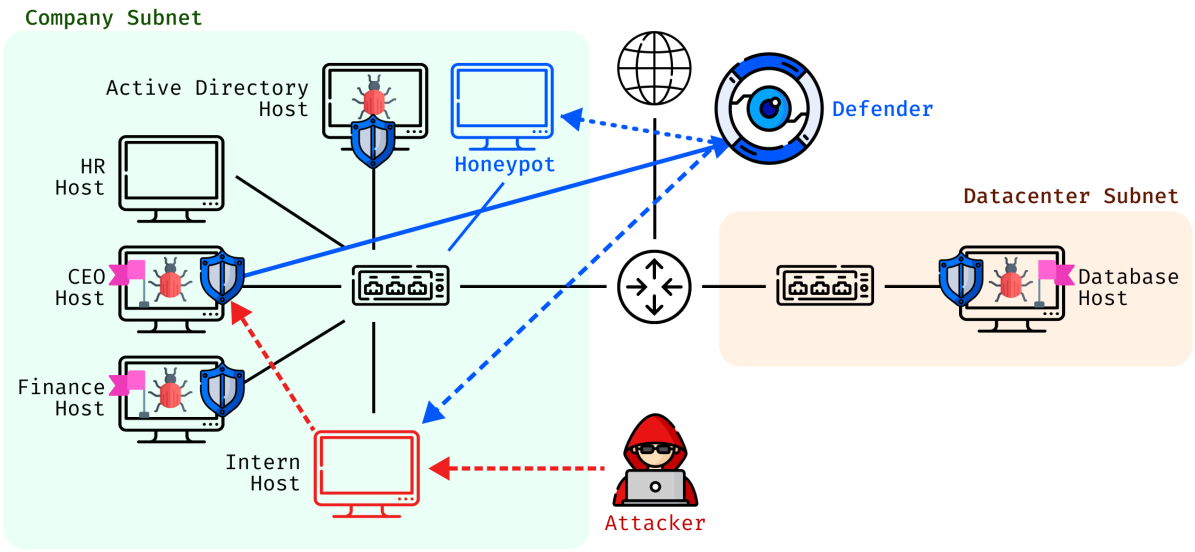

Figure 6.5: Experiment 3 Setup. The dotted blue arrow represents the defender responding to an alert by deploying a honeypot, represented by the blue host. Note that the defender will still restore the host interacting with the honeyservice to a clean state.

In this experiment, we build upon Experiment 2 by giving the defender the ability to deploy honeypots on the subnet from which it receives an alert of an attacker. Specifically, if the attacker receives an alert from a host on the `company` subnet, the defender will deploy a honeypot on that subnet, which the attacker will interact with and try to infect. Additionally, the defender also deploys honeyservices on the honeypot, meaning that if the attacker attempts to infect the honeypot by interacting with the honeyservice, the defender will be alerted and will restore the host that the attacker used for the attack. Importantly, we assume that the attacker *will* eventually interact with all honeypots that it can reach.

Hypothesis 3 *The attacker may capture all three flags, but will take significantly longer on average to do so than against the active defender. Additionally, we expect that, in most cases, the defender will prevent the attacker from capturing even a single flag.*

Experiment Outcome

Once again, as per the results shown in Tables 6.2 and 6.3 and Figure 6.6, the attacker captured all three flags only 5.6% of the time, which is a significant decrease from the active defender in Experiment 2. Additionally, the attacker took an average of 1258.9 seconds, which is nearly 400 seconds longer than any of the previous experiments, equating to a 45.1% increase over the active defender.

Furthermore, the defender prevented the attacker from capturing even a single flag in over 55% of the experiments, shutting the attacker down in an average of 284.45 seconds. We interpret this as an improvement over the average of 244.6 seconds with the active defender, since the attacker took longer to walk away with nothing, effectively wasting more of its resources. Additionally, the defender overall restored fewer real hosts than the active defender, as shown in Table 6.4, meaning that the defender detected the attacker earlier on in the attack chain, and thus deployed more honeypots to waste more of the attacker's time.

Finally, as shown in Table 6.5, the defender deployed two honeypots on average on the `company` subnet compared to the 0.12 average honeypots on the `datacenter` subnet, which meant that the defender narrowed down where the attacker was and attempted to waste more of its time by keeping it on the same subnet.

Since the attacker did not capture all three flags in most cases, and took significantly longer to do so than in the active defender, we can conclude that the defender's strategy was significantly more effective, and thus our hypothesis was correct.

Experiment 3 Takeaway

Smarter defensive strategies that changed and adapted with the attacker's actions further improved the efficacy of deception techniques, significantly increased the attacker's execution time, and in the vast majority of cases, prevented the attacker from achieving its goals entirely.

6.8 Experiment Results

Table 6.2: Overview of General Metrics for All Experiments

Metric	**Experiment 0**	**Experiment 1**	**Experiment 2**	**Experiment 3**
Total Trials	107	225	217	215
Average Setup Time	75.88 seconds	142.06 seconds	172.66 seconds	202.81 seconds
Average Execution Time	837.3 seconds	864.59 seconds	525.98 seconds	523.55 seconds

Table 6.2 shows an overview of the general metrics for all experiments, showing the total number of trials performed for each experiment in addition to the average **setup time**, the time it took to setup the experiment, and the average **execution time**, the time it took for the attacker to either finish capturing all three flags or for the defender to remove the attacker from the network. The average execution time does not seem to change significantly between experiments Experiment 2 and Experiment 3 due to the extremes in the execution time per total flags captured, as shown in Figure 6.6 and Table 6.3. Furthermore, the average setup time

increases between each experiment because we are performing additional setup steps for each experiment.

Table 6.3: Average Execution Time per Number of Flags Captured for All Experiments

Flags Captured	**Experiment 0**	**Experiment 1**	**Experiment 2**	**Experiment 3**
3	837.3 seconds	864.59 seconds	881.87 seconds	1258.9 seconds
2	-	-	774.23 seconds	927.24 seconds
1	-	-	455.62 seconds	594.58 seconds
0	-	-	244.6 seconds	284.45 seconds

Table 6.3 breaks down the average execution time per flag captured for each experiment. Note that the total flags captured is exclusive, meaning that, for example, if the attacker captured exactly three flags in a trial, it is not considered as having captured two flags.

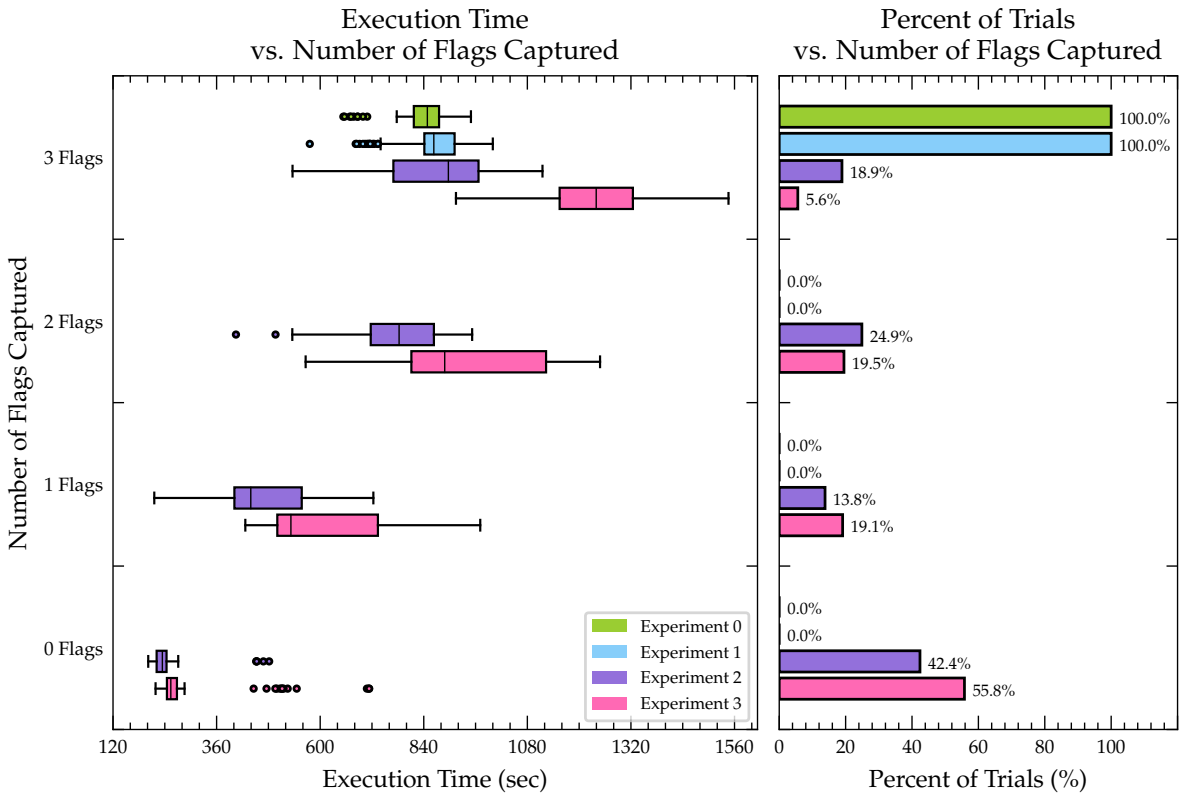

Figure 6.6: Results of All Experiments. Shows the time taken to capture a number of flags (left), and the percentage of trials that resulted in a number of flags captured (right) for each experiment and for each number of flags captured.

The graphs in Figure 6.6 show the results of all experiments. The graph on the left shows the time taken to capture a number of flags, indicating the median and interquartile range. The higher the execution time, and the lower the number of flags captured, the better it is for the defender. In other words, the defender wants the attacker to take as long as possible to capture as few flags as possible. Evidently, the dynamic defender (Experiment 3) is the most effective, taking significantly longer than all other experiments, followed by the active defender (Experiment 2), then the passive defender (Experiment 1), and finally the baseline (Experiment 0).

The graph on the right shows the percentage of trials that resulted in a certain number of flags captured. The higher the percentage of trials that resulted in a lower number of flags captured, the better it is for the defender, and the higher the percentage of trials that resulted in more flags captured, the worse it is. Specifically, the defender prefers that the percentage of trials resulting in three flags being captured is as low as possible while the percentage of trials resulting in zero flags being captured is as high as possible. Once again, the dynamic defender is the most effective, followed by the active defender, then the passive defender and the baseline.

Table 6.4: Average Restores Per Host for Experiments 2 and 3

Host	**Experiment 2**	**Experiment 3**
`activedir`	0.5 restores	0.31 restores
`intern`	1.19 restores	1.18 restores
`finance`	0.48 restores	0.31 restores
`database`	0.06 restores	0.12 restores
`ceo`	0.33 restores	0.24 restores

Table 6.4 shows the average number of times that the defenders of Experiments 2 and 3 restored a particular host. As we can see, both defenders restored the `intern` host the most, meaning it narrowed down where the attack was originating from. Additionally, on average, the dynamic defender in Experiment 3 performed less restores than the active defender in Experiment 2, meaning that the dynamic defender detected the attacker earlier on in the attack chain, and thus used other deception techniques (honeypots) to reduce the number of restores it had to perform.

Table 6.5: Average Honeypot Deployments Per Subnet for Experiment 3

Subnet	**Experiment 3**
`company`	2.04 honeypots
`datacenter`	0.12 honeypots

Table 6.5 shows the average number of honeypots deployed on each subnet. As we can see, the defender deployed two honeypots on average on the `company` subnet, which allowed the defender to narrow down where the attacker was and attempt to waste more of its time by keeping within the same subnet.

Chapter 7

Discussion

7.1 Limitations

In this section we briefly discuss some limitations of Perry that may not be of concern to the majority of use cases but we felt was important to mention. We believe that these limitations do not affect the validity of our findings nor the usefulness of the system in any significant way.

7.1.1 Emulated Attackers

In our experiments, we used emulated attackers to evaluate the effectiveness of our deception techniques. A natural limitation to emulation in general, and not specifically to Perry, is that the emulated attackers are not perfect representations of human attackers since they are not affected by the same things that humans are, nor are they as tenacious as human attackers can potentially be. However, we designed Perry in a way that allows us to easily extend the attacker to make it as complex as we need it to be with more sophisticated attacker patterns, thus emulating a human attacker as closely as possible. Despite this limitation, we believe that our findings are still valid since our attacker does indeed use real vulnerabilities and exploits to achieve its goals.

7.1.2 Simple Attacker

In our experiments, we used an attacker that was relatively simple and straightforward, and did not use any particularly advanced techniques to achieve its goals. This was done intentionally to demonstrate the effectiveness of our deception techniques, and to show that a simple attacker can indeed be thwarted by a well-designed deception strategy. However, more sophisticated attackers may be able to detect honeypots and avoid them altogether or even exploit them to gain access to the network. Furthermore, some malware can even detect if they are in a honeypot, and change their behavior accordingly [10], thus tainting any intelligence or data gathered from the honeypots they are on. We believe that this is an interesting avenue for future work, and that Perry can be used to explore this idea and to evaluate the efficacy of deception techniques against improved attackers.

7.1.3 Virtual Networks

This limitation is also not specific to PERRY, but rather to the use of virtual networks in general. Since we are using virtual networks residing on a cloud computing platform (OPENSTACK), we cannot faithfully emulate the network conditions that would be present in a physical network. For most use cases, this can be ignored or simply worked around by adding noise or latency to the virtual network, which PERRY does indeed accommodate. However, for some specific situations, such as for IoT and embedded devices, that require real-time communication and near-zero latency, this *may* be a significant limitation, but we are not certain of this. We leave the task of evaluating this limitation for future work.

Chapter 8

Future Work

Throughout the development of PERRY, we came up with several intriguing ideas that were not in the scope of this thesis. In this section we will discuss some of those ideas and how they can be used to improve and build upon PERRY.

8.1 Red Patching

One of the challenges we faced during the development of the attacker capabilities in PERRY was finding realistic vulnerabilities and implementing or finding reliable exploits for the attacker to use. During the later development phase of this project, we established communication with the researchers at IBM responsible for SYSFLOW, who introduced the idea of HoneyPatches [33], in addition to a novel system they are working on that would allow us to bypass the need to write **Proof of Exploits** for the attacker, which are evidences that a vulnerability can be exploited, typically done through demonstration of exploit without causing harm. This would also eliminate requirement of having real vulnerabilities on the hosts in the deployment instance and works by creating process instrumented with a backdoor to allow us to storyboard events rather than rely on inconsistent exploits throughout the attacker's infection process.

For example, instead of placing a real vulnerability on a real FTP service, we would instead have a real FTP service instrumented with their system that allows the attacker to achieve the same effect as if they had exploited the real vulnerability with a real exploit.

While this is potentially a very useful and interesting idea, it is unclear how this would affect the realism of the attacker or the efficacy of deception strategies. We suspect that, if done correctly with careful consideration for the types of services being instrumented, this would not have significant impact, but that is something that is not within the scope of this thesis, and would be a potential avenue for future work.

8.2 Game Theory and Machine Learning

As mentioned in Section 4.2, we designed PERRY similar to game theoretic models of security games, in which two players, typically an attacker and a defender, interact strategically in order to achieve their goals [25]. In this thesis, we did not explore the game theoretic aspect of PERRY, but we believe that it would be a strong potential avenue for future work. In particular, we

believe that it would be particularly useful to explore the Nash Equilibria of the system, and how the defender can use this information to determine the optimal strategy to use in order to maximize their payoff. Additionally, we believe that it would be compelling to explore the use of machine learning in PERRY to allow the defender to learn the attacker's behavior and adapt to it accordingly, and to create even more realistic attackers.

8.3 Refining PERRY

We designed PERRY to be flexible and extensible, however due to time constraints we were not able to take full advantage of this, and only implemented the capabilities and strategies necessary for our experiments. However it would be highly beneficial to explore the full extent of PERRY's features by implementing more deception abilities, such as honeyfiles, honeyusers, and honeytokens. Additionally, it would be useful to test the efficacy of deception strategies against more sophisticated attackers, such as those that can detect and avoid honeypots, or even exploit them.

Futhermore, it would be interesting to see the goal of the defender and attacker be implemented in different ways, such as the defender gathering intelligence instead of trying to remove the attacker from the network, or the attacker trying to actively destroy the network instead of gathering flags. Finally, it would also be interesting to explore the use of different network topologies and see how that affects the efficacy of the defender's strategies. We believe that there is a lot of potential for future work in this area and that PERRY can be used as a platform to explore these ideas.

Chapter 9

Conclusion

Deception is a powerful tool that is effective at preventing attackers from achieving their goals of infiltrating a network and gaining access to its resources. We have taken inspiration from real-world military tactics and applied them to the network security domain, and we have seen promising results that indicate that deception may be pivotal in reversing the asymmetry between attackers and defenders within the space of cybersecurity.

There is a growing interest in deception as a defensive technique, yet we have noticed that there was a lack of a comprehensive framework or platform for evaluating their effectiveness. In this master's thesis, we proposed Perry, a flexible, extensible, realistic, and automated platform that can be used to evaluate the efficacy of deception techniques against various attackers across a range of different scenarios. We described the high-level system abstraction before delving deep into the implementation details of our platform, showing that the system is sound and that it works. We then demonstrated the effectiveness of Perry by using it to evaluate the efficacy of several deception techniques from prior work against an attacker using real exploits and vulnerabilities. We ran several experiments, each with a different defender that used different strategies and capabilities to determine how effective each one was at preventing attacks. We observed that a combination of deception techniques, telemetry, and intelligent strategies is effective at significantly slowing down attackers and preventing them from achieving their goals. Finally, we discussed the limitations of Perry and the potential avenues for future work. Taking advantage of Perry's foundation, future researchers can build upon our platform and extend it to realistically evaluate current and future deception techniques against improved, sophisticated attackers and in more complex scenarios.

Bibliography

[1] S. Tzu, *The Art of War*. 2010.

[2] J. J. Yuill, "Defensive computer-security deception operations: Processes, principles and techniques", Ph.D. Dissertation, North Carolina State University, 2010. [Online]. Available: http://www.lib.ncsu.edu/resolver/1840.16/5648.

[3] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley Longman Publishing Co., Inc., 2002, ISBN: 0321108957.

[4] N. Vaideeswaran. "What are honeytokens?: Crowdstrike". (2023), [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/identity-security/honeytokens/.

[5] J. Yuill, M. Zappe, D. Denning, and F. Feer, "Honeyfiles: Deceptive files for intrusion detection", in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, 2004, pp. 116–122. DOI: 10.1109/IAW.2004.1437806.

[6] L. Spitzner, "Honeypots: Catching the insider threat", in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, 2003, pp. 170–179. DOI: 10.1109/CSAC.2003.1254322.

[7] E. S. Pilli, R. Joshi, and R. Niyogi, "Network forensic frameworks: Survey and research challenges", *Digital Investigation*, vol. 7, no. 1, pp. 14–27, 2010, ISSN: 1742-2876. DOI: https://doi.org/10.1016/j.diin.2010.02.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287610000113.

[8] O. U. Oluoha, T. S. Yange, G. E. Okereke, and F. S. Bakpo, *Cutting edge trends in deception based intrusion detection systems — a survey*, 2021. DOI: 10.4236/jis.2021.124014. [Online]. Available: http://dx.doi.org/10.4236/jis.2021.124014.

[9] E. Gartzke and J. R. Lindsay, "Weaving tangled webs: Offense, defense, and deception in cyberspace", *Security Studies*, vol. 24, no. 2, pp. 316–348, 2015. DOI: 10.1080/09636412.2015.1038188. eprint: https://doi.org/10.1080/09636412.2015.1038188. [Online]. Available: https://doi.org/10.1080/09636412.2015.1038188.

[10] M. H. Almeshekah, E. H. Spafford, and M. J. Atallah, "Improving security using deception", *Center for Education and Research Information Assurance and Security, Purdue University, Tech. Rep. CERIAS Tech Report*, vol. 13, p. 2013, 2013.

[11] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective", *ACM Comput. Surv.*, vol. 51, no. 4, 2018, ISSN: 0360-0300. DOI: 10.1145/3214305. [Online]. Available: https://doi.org/10.1145/3214305.

[12] T. Shade, A. Rogers, K. Ferguson-Walter, S. B. Elsen, D. Fayette, and K. E. Heckman, "The moonraker study: An experimental evaluation of host-based deception.", in *HICSS*, 2020, pp. 1–10.

[13] C. Wang and Z. Lu, "Cyber deception: Overview and the road ahead", *IEEE & Security Privacy*, vol. 16, no. 2, pp. 80–85, 2018. DOI: 10.1109/MSP.2018.1870866.

[14] M. Ben Salem and S. J. Stolfo, "Decoy document deployment for effective masquerade attack detection", in *Detection of Intrusions and Malware, and Vulnerability Assessment*, T. Holz and H. Bos, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 35–54, ISBN: 978-3-642-22424-9.

[15] J. Voris, J. Jermyn, N. Boggs, and S. Stolfo, "Fox in the trap: Thwarting masqueraders via automated decoy document deployment", in *Proceedings of the Eighth European Workshop on System Security*, ser. EuroSec '15, Bordeaux, France: Association for Computing Machinery, 2015, ISBN: 9781450334792. DOI: 10.1145/2751323.2751326. [Online]. Available: https://doi.org/10.1145/2751323.2751326.

[16] P. Aggarwal, A. Gautam, V. Agarwal, C. Gonzalez, and V. Dutt, "Hackit: A human-in-the-loop simulation tool for realistic cyber deception experiments", in *Advances in Human Factors in Cybersecurity*, T. Ahram and W. Karwowski, Eds., Cham: Springer International Publishing, 2020, pp. 109–121, ISBN: 978-3-030-20488-4.

[17] P. Aggarwal, M. Gutierrez, C. D. Kiekintveld, B. Bošanský, and C. Gonzalez, "Evaluating adaptive deception strategies for cyber defense with human adversaries", in *Game Theory and Machine Learning for Cyber Security*. 2021, pp. 77–96. DOI: 10.1002/9781119723950.ch5.

[18] K. J. Ferguson-Walter, M. M. Major, C. K. Johnson, and D. H. Muhleman, "Examining the efficacy of decoy-based and psychological cyber deception", in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Aug. 2021, pp. 1127–1144, ISBN: 978-1-939133-24-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/ferguson-walter.

[19] T. E. Carroll and D. Grosu, "A game theoretic investigation of deception in network security", *Security and Communication Networks*, vol. 4, no. 10, pp. 1162–1172, 2011. DOI: https://doi.org/10.1002/sec.242. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.242. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.242.

[20] F. Cohen and D. Koike, "Feature: Leading attackers through attack graphs with deceptions", *Comput. Secur.*, vol. 22, no. 5, pp. 402–411, 2003, ISSN: 0167-4048. DOI: 10.1016/S0167-4048(03)00506-6. [Online]. Available: https://doi.org/10.1016/S0167-4048(03)00506-6.

[21] A. H. Anwar and C. Kamhoua, "Game theory on attack graph for cyber deception", in *Decision and Game Theory for Security: 11th International Conference, GameSec 2020, College Park, MD, USA, October 28–30, 2020, Proceedings*, College Park, MD, USA: Springer-Verlag, 2020, pp. 445–456, ISBN: 978-3-030-64792-6. DOI: 10.1007/978-3-030-64793-3_24. [Online]. Available: https://doi.org/10.1007/978-3-030-64793-3_24.

[22] C. Baillie, M. Standen, J. Schwartz, M. Docking, D. Bowman, and J. Kim, *Cyborg: An autonomous cyber operations research gym*, 2020. arXiv: 2002.10667 [cs.CR].

[23] L. Li, R. Fayad, and A. Taylor, *Cygil: A cyber gym for training autonomous agents over emulated network systems*, 2021. arXiv: 2109.03331 [cs.CR].

[24] D. Reti, D. Fraunholz, K. Elzer, D. Schneider, and H. D. Schotten, "Evaluating deception and moving target defense with network attack simulation", in *Proceedings of the 9th ACM*

Workshop on Moving Target Defense, ACM, Nov. 2022. DOI: 10.1145/3560828.3564006. [Online]. Available: https://doi.org/10.1145%2F3560828.3564006.

[25] S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu, "A survey of game theory as applied to network security", in *Proceedings of the 2010 43rd Hawaii International Conference on System Sciences*, ser. HICSS '10, USA: IEEE Computer Society, 2010, pp. 1–10, ISBN: 9780769538693.

[26] M. Howard, "Terraform – automating infrastructure as a service", 2022. arXiv: 2205.10676 [cs.SE].

[27] Y. Brikman, *Terraform: Up and Running*. " O'Reilly Media, Inc.", 2022.

[28] O. Sefraoui, M. Aissaoui, M. Eleuldj, *et al.*, "Openstack: Toward an open-source solution for cloud computing", *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[29] L. Hochstein and R. Moser, *Ansible: Up and Running: Automating configuration management and deployment the easy way*. " O'Reilly Media, Inc.", 2017.

[30] R. Alford, D. Lawrence, and M. Kouremetis, "Caldera: A scalable, automated adversary emulation platform", *MITRE: Bedford, MA, USA*, 2022. [Online]. Available: https://github.com/mitre/caldera.

[31] MITRE. "Caldera documentation". (2023), [Online]. Available: https://caldera.readthedocs.io/en/latest/Getting-started.html (visited on 08/20/2023).

[32] F. Araujo and T. Taylor, "Sysflow: Scalable system telemetry for improved security analytics", in *IEEE International Conference on Big Data*, 2020.

[33] F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser, "From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation", in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14, Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 942–953, ISBN: 9781450329576. DOI: 10.1145/2660267.2660329. [Online]. Available: https://doi.org/10.1145/2660267.2660329.

[34] B. Elasticsearch, "Elasticsearch", *software], version*, vol. 6, no. 1, 2018.

[35] *CVE-2011-2523*, Available from MITRE, CVE-ID CVE-2011-2523, 2011. [Online]. Available: https://www.cve.org/CVERecord?id=CVE-2011-2523.