Adaptive Cyber Deception Game Implementation and Deployment

Adam Hunt, Pau Balcells Sanchez
Carnegie Mellon University
Pittsburgh, PA, US

Abstract

Attacks on networks, as well as the expansion of their targets, have become a growing problem. As we become more dependent on technology, our networks have had to adapt and grow to keep up with the ever-increasing demand. As a consequence, the surface that is susceptible to cyber-attacks has grown at an alarming rate. Currently, there is a shortage of resources and manpower to keep up with the maintenance, defense, and mitigation efforts needed to maintain the security of these growing networks. Cyber deception techniques constitute one promising method of increasing defender advantage. In this work, we describe our contributions to cyber deception research of deploying a real-world vulnerable network and creating the infrastructure needed for an AI agent to interact with it using network monitoring and a set of cyber deception techniques. We propose a new model that can be used with real-world networks with multiple attackers. We also describe heuristic attacker and defender strategies in this environment and offer discussion on the future work of training a reinforcement learning (RL) policy to implement an adaptive defense strategy.

aws deployment, c2 architecture, translation layer (graph¡-¿real-world), refined attack graph definition (advantage: allows for multi-attacker)

Introduction

With the increase in our dependence on technology, the importance of our networks has significantly increased: we currently not only rely on our networks to store more critical and sensible information, but we also depend on them to carry out daily tasks such as remote work, shopping online, or even keeping track of our finances. This over-dependence on technology and the networks that support it brings up two major problems:

- **Too Many Networks to Keep Track Of:** The growth in network dependence and the lack of professionals in the area of cybersecurity have made it impossible for us to maintain these networks and keep them secured manually. Security is a never-ending cycle in which security professionals have to keep scanning infrastructure to find and mitigate all vulnerabilities before they become exploited. It is a constant race between security engineers and malicious attackers to see who can find and act on a vulnerability first. The absence of professionals in the cybersecurity field and the increasing capabilities and creativity that attackers are equipped with to break into infrastructure presents a huge problem: security engineers cannot keep up with the large networks they have to defend at a fast enough pace. Unfortunately, this compromises and diminishes the security of all technological infrastructure around the world.

- **Wider Attack Surface:** The increase in network infrastructure, combined with the lack of professionals that can properly maintain it and keep it safe, has made it so that the attack surface that attackers can exploit to infiltrate a network has grown. This, again, makes it easier for attackers to find vulnerabilities to exploit and harder for security professionals to mitigate these vulnerabilities in time. Moreover, in order to adequately defend a network when it has been breached by attackers, defenders have to act quickly and flawlessly in an effort to minimize damages. The analysis required to identify, confirm, and defend against network intruders is not fast enough.

In an effort to address these challenges, there has been a recent line of research with the goal of developing Autonomous Defenses against cyber attacks. In this work, we build off of the work of Yinuo et al. in [1], which describes an adaptive cyber deception game and a proposed RL-based defender to play it. Our work centers around the deployment of this method to apply it in a real-world network. In this paper, we describe our deployment of a real-world vulnerable network and our creation of the infrastructure needed for an AI agent to interact with it using network monitoring and a set of cyber deception techniques. We propose a new model that can be used with real-world networks with multiple attackers. We also describe heuristic attacker and defender strategies in this environment and offer discussion on the future work of training a reinforcement learning (RL) policy to implement an adaptive defense strategy.

Related Work

Attack Graph

One of the key components behind our project is the use of attack graphs [4]. With the complexity and scale that networks have nowadays, attack graphs are a versatile modeling tool that helps resolve a wide array of security challenges. Attack graphs are a great resource to translate the attack surface of a network, as well as to represent the evolution of the state of a network to RL models [9, 5]. There has even been research to use attack graphs to detect previously undiscovered vulnerabilities with a network [3]. For this project, we focus on the basic representation capabilities that attack graphs have to translate the network state to our defensive agent. Each node in an attack graph represents a different attack state in the network, while each edge represents a different attack the attacker can exploit within the network.

Reinforcement Learning for Cyber Defense

Reinforcement Learning has been shown to be highly effective in training cyber defensive agents that successfully defend dynamic network environments, even in cases where there is limited prior knowledge about the environment [6, 8, 2]. The models are trained to take sequential defensive and deceptive actions within the environment to protect and defend it from attackers. The addition of Deep Learning and Proximal Policy Optimization (PPO) has been shown to improve the defensive capabilities of the defensive models and widen the possible applications of these agents within cyber security [7]. More specifically, PPO has been shown to perform successfully in multi-agent games, allowing these defensive agents to protect the network against multiple simultaneous attackers[11]. We have based this project on the foundation established by [1], which uses a combination of RL and PPO to train a defensive agent that autonomously defends a network against intruders.

Game Model

For this project, we use an attack graph model of the network environment where one or more attackers try to infiltrate the network and retrieve as much information stored in the network as possible while the defender tries to protect its data and create opportunities to gain threat intelligence about the attacker. This problem model consists of the following pieces.

Attack Graph:

As previously mentioned, our model relies on the existence of an attack graph that represents all the possible states of each host within the network. The specific attack graph used for our project can be found in Figure 2. The elements of the network are the following:

Nodes Each node corresponds to an attack stage on one of the hosts. In accordance with the commonly accepted cyber kill chain [10], we define three nodes per machine in the network: 1. *foothold established*: When an attacker has gained low privileged (e.g., standard user) command execution capabilities on a given machine. 2. *privilege escalated*: When an attacker has gained high privilege (e.g., administrator or root) command execution capabilities on a given machine. 3. *data exfiltrated*: When an attacker has taken valuable data from a machine.

Each node is assigned a value representing the asset in that state. Node value is based on the data stored on a machine. It is only non-zero for data exfiltration nodes.

Edges The edges of the attack graph represent the actions needed to get to a given attack state, i.e., gaining a foothold, escalating privileges, or exfiltrating data. Each edge is assigned a probability representing the chances of an attacker successfully traversing the edge. The probability of an attacker successfully transitioning across an edge of the attack graph given it tries is given to the defender at the beginning of the game. It is based on vulnerability scans of the network and its hosts. It can be learned by the attacker through reconnaissance, and the perceived probability can be altered by the defender through deceptive actions.

Attacker's Action Space:

The attacker's action space consists of:

1. *Gain Foothold:* The attacker can gain a foothold on a host by exploiting a vulnerability.
2. *Privilege Escalation:* Once the attacker has established itself in a host, its main goal will be to gain root access to that host, which will give it full privilege over it.
3. *Data Exfiltration:* Once the attacker has gained root access, they can finally exfiltrate any data they are interested in.

These actions are abstracted in the attack graph as the traversal through edges. Each node represents an attack stage on one of the hosts, while each edge will represent the different actions that the attacker could take from those positions. For example, if the attacker has established a foothold in a machine, it will currently sit on that node. The edges coming off that node are the attacks it will have to perform to get to a privilege escalation state or gain a foothold on another host. An attacker may be at more than one node at once, in which case its set of actions would be the set of all edges from the nodes at which it exists. Equally, multiple concurrent attackers can be on the attack graph in various positions; the state of the attack graph simply shows which nodes currently have at least one attacker on them.

Attacker's Observation

The attacker is aware of its current locations in the network. Through enumeration and reconnaissance techniques, it can also observe the outgoing edges from these nodes. From this information, it can also make an estimate of the value associated with a node. For example, if an attacker scans a host and sees that it has a database server running, it might believe that this is a high-value host due to the data it potentially contains.

The attacker's observations are influenced by the deceptive tactics of the defender. For example, if the defender

starts an SSH honey service on a machine, the attacker will believe that SSH is a valid method to connect to this machine.

Defender's Action Space:

The defender has three deceptive techniques it can use. At any given time it can start or stop any of these techniques on any host on the network.

1. *Create Fake Foothold:* The defender starts an SSH honey service with an easily guessed username and password on the host. If an attacker successfully bruteforces the credentials and connects to the service, it will be put in a limited shell where it is isolated from the rest of the system and its actions and file uploads are logged and stored.
2. *Create Fake Privilege Escalation:* Similarly to the above action, the defender can trick the attacker into believing that it has found a way to escalate privileges within a host by creating a root-owned executable with the *Set User ID (SUID)* bit set, meaning the executable will show up in common enumeration scripts as a method for privilege escalation. Instead of providing a route to root, this executable simply outputs an enigmatic message designed to confuse the attacker and logs the activity.
3. *Create Fake Data:* The defender creates a file on the host that resembles the true data of value on the system, but without any of the actual data. When the attacker finds this data, it triggers a sensor by reading it and moves on to the next machine, believing it has already retrieved the real data.

Defender's Observation:

The defender's observation consists of two parts. First, the attack graph itself, with its nodes and edges, including both real and fake edges. Second, the defender can see attacker positions over the last five minutes. Attacker positions are defined by a node on the graph (i.e., host and attack stage) and are each associated with a timestamp. These are collected by the distributed sensors of the command and control infrastructure and pick up attacker actions as they attempt to cross real or fake edges (i.e., attempt exploits of real or fake vulnerabilities and data exfiltration).

Probabilistic Knowledge of Attacker Position The attacker's position is unknown until it is detected by a defender sensor. Once this happens, the position is known with certainty to be around the fake edge represented by that sensor. As time passes and the attacker has more opportunity to transition away from where it was previously detected, the defender's certainty decreases.

End of Game

In addition to detecting the attacker, the defender is also interested in engaging with the attacker. To support this, the game does not end when the attacker is discovered; rather, the defender can decide to end the game at any moment after it initially detects the attacker. This would be done by blocking the IP of the attacker. This ends one instance of the game. At this point, final scores can be calculated.

Attacker's Reward

The attacker gets points for exfiltrating data for the network. Each machine has a point-value associated with its data. When the attacker exfiltrates this data, it gains these points and the defender loses an equivalent amount.

Defender's Reward

In addition to the penalty for allowing data to be stolen, the defender's reward scheme is designed to incentivize interacting with the attacker to generate threat intelligence. The defender is rewarded for the number of observations it captures of the attacker before the end of the game.

Evaluation

To allow for model training and evaluation, we designed and deployed an AWS network injected with several vulnerabilities. We chose this network scenario because it could be successfully scaled up and applied to real-world network infrastructures, such as a company's technological infrastructure.

Network Topology As seen in Figure 1, the network consists of three subnetworks. The first subnetwork is public facing and contains two *WordPress* servers accessible through the Internet. These servers are the two entry nodes to the network and will be the first step in any attempts to intrude on the rest of the network. The second subnetwork, a private-facing one, contains the NTP and DB servers. This subnetwork will be the second stage of any attack, as the only way to jump from the public-facing subnetwork to the rest of the network is through either the NTP or DB servers due to the company's firewall rules. The third subnetwork, also a private-facing one, contains an internal web server and three PCs with access to it. Additionally, there is a fourth private subnetwork (see Figure 3) which contains the Controller equipped with the Defensive model. This does not accept any externally initiated traffic, so only the Controller can establish connections with all the other devices in the network. The attacker cannot reach it; only the other three subnetworks are meant to be reached.

Vulnerabilities Table 1 contains a list of the vulnerabilities that we used to infect the network. These vulnerabilities allow the attacker to gain a foothold on a host and escalate to root privileges within them. We chose these vulnerabilities as being representative of significant classes of vulnerabilities that exist in modern networks, such as insecure configurations, weak passwords, outdated internal tools, and failures to patch recently released vulnerabilities. The table shows a brief analysis of these vulnerabilities. The difficulty level shown is relative to the other vulnerabilities and attempts to qualitatively show the complexity of detecting and exploiting this vulnerability. The *Number of Steps* and *Approximate Lines of Code* metrics are based on the implementations of the exploits for these vulnerabilities that were used in our testing. *Noise* shows how easy an exploit of the vulnerability would be to detect, the ones marked high being those which generate network traffic while the others are only visible from the local system they are used on. The

Table 1: Vulnerabilities

Vulnerability	Difficulty	Number of Steps	Approximate Lines of Code	Noise	CVSS Score	Host Infected
WordPress Backup Migration	Medium	1	300	High	9.8	Public Servers
Weak SSH Credentials	Medium	2	2	High	N/A	NTP, DB, WEB servers
Weak Sudo Version	Easy	1	1	Low	N/A	NTP, DB, WEB servers
World-writable script executed by cronjob as root	Medium	3	15	Low	N/A	Public Servers

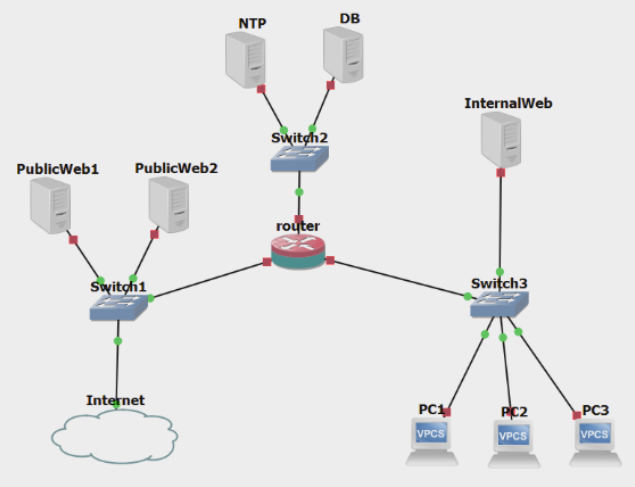

Figure 1: Network Diagram

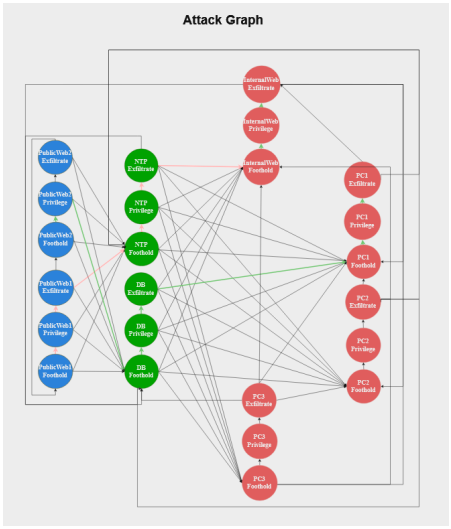

Figure 2: Attack Graph

CVSS Score is only given for the first vulnerability and represents the severity of the vulnerability on a scale out of ten. Finally, the *Host Infected* column details which systems on our network had the vulnerability.

Attack Graph

Figure 2 shows the attack graph we created for our simulation, which the model will use to observe the state of the network and the attacker in it. The blue nodes represent the different states of the Public Servers, the green ones are the states of the NTP and DB servers, and the red ones are those of the Web servers and PCs. The directed edges represent the different actions that the attacker can take from the nodes that it holds. The green and red arrows show two different plausible attacking paths as an example.

Technical Foundations

Cloud deployment To deploy the network and run the simulation environment, we chose AWS. To deploy it, we created a VPC containing the four subnets previously mentioned. In order to provide internet access to the private subnets without them being directly accessible, we equipped the three private subnets with a NAT gateway. Within each subnet, we launched an *Ubuntu Server 24.04 LTS (HVM), SSD Volume Type* Amazon Machine Image instance equipped with *t3.medium* with 2 virtual CPUs with 4 GiB of RAM and 16 GiB of *gp3* storage for each one of the hosts of the network. All these instances can be reached through ssh for setting the network up and equipping each one with the necessary software and their corresponding vulnerabilities. On top of that, each instance contains its own firewall set through *AWS Security Groups*. Each instance only accepts traffic from the subnets to which it is directly connected, as well as the Controller subnet, only in the necessary ports required for their purpose. Moreover, all instances accept TCP traffic on port 17737, which is used to establish communication between the Controller and the rest of the network.

Command and Control The *Command and Control (C2)* architecture is one of the main foundations behind this

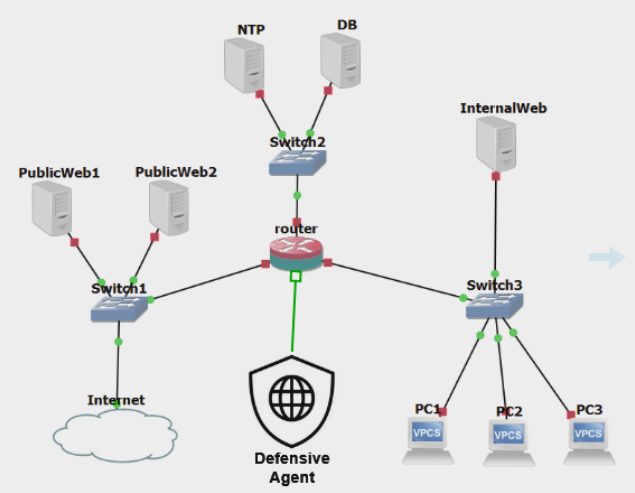

Figure 3: Command Control System Diagram

project. It allows the Controller to interact with and gather information about the network. It is implemented in Python and makes use of gRPC and Protocol Buffers for communication. Figure 3 shows a system architecture diagram where the Command Control sits.

- **Network Interaction:** Through *C2*, the defender can:
  - Create an executable file with *suid* bit set to waste attacker time trying to escalate privileges.
  - Create fake files in a host to trick the attacker into thinking it is ex-filtrating sensible data.
  - Spin up a Cowrie SSH Service (honeypot) to trap the attacker.
- **Information Gathering:** The *Command Control* system manages a range of sensors on each system that detects attempts to exploit real and fake vulnerabilities in the system. This is done primarily by monitoring system logs and using Linux's *inotify* system to detect when a given file is read or modified. This sensor data is timestamped and gathered when the agent chooses to make an observation.

Defender Agent Interface The C2 system has a graph abstraction layer that translates high-level graph actions into specific, real-world actions that can be taken on the network. The agent's interface makes use of functions to enumerate, add, and remove edges as well as getting attacker positions.

Experiment

Autonomous Attacker To measure the effectiveness of the defender strategies and provide an environment in which the RL-based defender could learn, we created an autonomous attacker in the form of a computer worm, specially tuned to the vulnerabilities of our deployment. The worm follows a predetermined path through the network, using exploits to propagate itself from one host to another, starting from a public-facing web server, pivoting through the database server, and then infecting all machines in the internal user subnetwork. Once on a machine in a low-privileged context, the worm attempts to carry out one of a number of exploits in order to execute itself in a higher-privileged context before exfiltrating the machine's data.

Without any defender intervention, the worm will fully exploit all machines in its path, exfiltrating all data of value in the process. However, following the original model of a weak attacker from [1], a deceptive tactic will misguide the attacker and prevent it from taking its intended action. For example, if the defender places fake valuable data on the system, the attacker will exfiltrate this and move on, thinking it has gotten what it came for. If an attacker fails to escalate privilege because it has been deceived with a fake privilege escalation vulnerability, it will move on to the next system without gaining root privileges. Finally, if the defender places a fake foothold on a machine currently implemented as an SSH honey service, the attacker will get lost attempting to run in this fake shell and will not be able to continue its attack.

Heuristic Defender To protect against this heuristic attack, whether by the worm or a hands-on keyboard attacker, we implemented a command-line control interface that allows a defender to manually deploy deceptive actions. We also implemented a network monitor that integrates with the C2 system to show the defender the attacker's actions on the network. With these two, we implemented a heuristic strategy that allowed the attacker to exploit the public server fully but protected the valuable data on the database server from exfiltration and caught the attacker in an SSH honey service as it tried to pivot to the internal user subnetwork.

To further optimize the defender's advantage, a stronger heuristic strategy would take a similar approach to letting the attacker explore the network in a controlled fashion, but would immediately deploy fake valuable data on all systems upon detecting an attacker in the network (or perhaps even prior to this point) to prevent the attacker from gaining points through exfiltration.

Reinforcement learning-based Defender Using the system we have developed, an RL-based defender strategy will be able to be trained. We plan on testing a PPO with Long Short-Term Memory (LSTM) strategy to learn a policy on the network due to this approach's success under the original model from [1]. We will implement action masking to reduce the number of possible actions to choose from, only considering those near an attacker location. Because the new model supports multiple attackers, we will also test how this approach fairs under these conditions.

Results

With the assumption of a weak attacker (i.e., one that is fully deceived by defender actions), even a heuristic defender is able to dominate by deploying counterfeit data on all hosts on the network. This prevents the attacker from gaining any points, regardless of machines compromised and time spent on the network.

Beyond protecting data, there is value to the defender in gaining threat intelligence about its attackers. Conversely,

the attacker wants to keep its tools, tactics, and procedures secret to maintain their effectiveness against this and other targets. Qualitatively, the heuristic defender strategy does a good job of allowing the attacker, with its predetermined route, to take actions on the network, limiting its ability to take real actions as it progresses further into the network. The SSH honey service was especially effective in gathering observations on the attacker without letting it get to the valuable parts of the network.

Each game with the heuristic attacker worm takes between one and two minutes, the bulk of this time being due to an exploit that modifies a script that is automatically run every minute; thus, the attacker waits one minute after making this modification to be sure the script has been executed.

Discussion

By taking the abstract model from [1] and attempting to deploy it, we discovered the ways in which the model would need to be modified to work in a real-world network. In addition, we contribute a system for connecting the modified adaptive defender agent to a deployed network to allow it to learn and defend in the real world with real vulnerabilities and defensive tactics. Our system can be adapted to run on a wide variety of networks and can be extended to use a greater variety of deceptive techniques according to the environment in which it will be deployed.

The results show promise for deploying adaptive defense mechanisms such as from [1] in real-world networks, which could have a substantial positive impact on society by giving the advantage back to defenders. By going beyond simple defensive tactics and actively engaging an attacker with deceptive techniques, defenders can not only protect their networks from the immediate dangers of an attack, but can gather valuable information that can be shared and used against attackers.

There are a number of items that remain as future work due to some unforeseen complexities and changes in project goals. Among these are the automatic deployment of the network, including honeypots, and the training of a defensive agent. While this training has not yet been done, our network deployment, C2 system with its translation layer, and heuristic attacker will allow effectively facilitate this training after a few modifications to the agent's code to account for the updated model.

Acknowledgements

This project exists under the umbrella of Yinuo Du's ongoing research in Adaptive Cyber Deception and was conducted with her incredible guidance and mentorship and with the advisement of the amazing Dr. Fei Fang.

References

[1] Yinuo Du et al. "Learning to play an adaptive cyber deception game". In: 6 (2022).

[2] Ming Feng and Hao Xu. "Deep reinforecement learning based optimal defense for cyber-physical system in presence of unknown cyber-attack". In: (2017), pp. 1–8. DOI: 10.1109/SSCI.2017.8285298.

[3] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. "Practical Attack Graph Generation for Network Defense". In: (2006), pp. 121–130. DOI: 10.1109/ACSAC.2006.39.

[4] Igor Kotenko and Mikhail Stepashkin. "Attack Graph Based Evaluation of Network Security". In: (2006). Ed. by Herbert Leitold and Evangelos P. Markatos, pp. 216–227.

[5] Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. "A review of attack graph and attack tree visual syntax in cyber security". In: *Computer Science Review* 35 (2020), p. 100219. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2019.100219. URL: https://www.sciencedirect.com/science/article/pii/S1574013719300772.

[6] Li Li, Raed Fayad, and Adrian Taylor. "CyGIL: A Cyber Gym for Training Autonomous Agents over Emulated Network Systems". In: (2021).

[7] Thanh Thi Nguyen and Vijay Janapa Reddi. "Deep Reinforcement Learning for Cyber Security". In: *IEEE Transactions on Neural Networks and Learning Systems* 34.8 (2023), pp. 3779–3795. DOI: 10.1109/TNNLS.2021.3121870.

[8] Aritran Piplai et al. "Knowledge Guided Two-player Reinforcement Learning for Cyber Attacks and Defenses". In: (2022), pp. 1342–1349. DOI: 10.1109/ICMLA55696.2022.00213.

[9] Lingyu Wang et al. "An Attack Graph-Based Probabilistic Security Metric". In: (2008). Ed. by Vijay Atluri, pp. 283–296.

[10] Tarun Yadav and Arvind Mallari Rao. "Technical aspects of cyber kill chain". In: *Security in Computing and Communications: Third International Symposium, SSCC 2015, Kochi, India, August 10-13, 2015. Proceedings 3*. Springer. 2015, pp. 438–452.

[11] Chao Yu et al. "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games". In: 35 (2022). Ed. by S. Koyejo et al., pp. 24611–24624. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/9c1535a02f0ce079433344e14d910597-Paper-Datasets_and_Benchmarks.pdf.