HONEYPOT ALLOCATION FOR NETWORK HARDENING IN INDUSTRIAL CONTROL SYSTEMS

by

PASCAL DUNCAN LEK HOU U
URN: [redacted]

A dissertation submitted in partial fulfilment of the
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May 2025

Department of Computer Science
University of Surrey
Guildford GU2 7XH

Supervised by: Martín Barrère Cambrun

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Pascal Duncan Lek Hou U
May 2025

Abstract

This project studies the problem of network hardening in industrial control systems (ICSs) given an administrator having limited resources to allocate. ICSs are a frequent target of attacks because of their importance to infrastructure and their notoriety for utilising legacy systems. Honeypots (HPs) have been used as a form of intrusion detection system which can detect a vast swathe of possible attacks. However, given that the administrator can only afford to have a select number HPs within a system, an allocation technique is required. I propose a game-theoretic framework, which models a defender and attacker against each other, both attempting to maximise their stated goals. This is done through a reward function that rewards the defender for successfully intercepting an attack, but penalises it for non-interceptions. The attacker's reward function is the direct inverse of this and as such this game is what is called a Zero-Sum game. The creation and preparation of the game-action space is done by utilising a novel graph generator and reducing it through creating what are known as CAGs so that the complexity of the action space is reasonable for large network sizes. Analysis is conducted on various graph topology sizes in evaluating the performance and scalability of the model, including varying key parameters and their effects on the game outcome.

Acknowledgements

First and foremost, I give my deepest thanks to my family who have always supported, inspired and pushed me in my pursuit of a bachelor's degree.

To the many peers I have had the privilege of spending countless hours with in the Ada Lovelace Lab. Many of whom undoubtedly have influenced my way of thinking throughout the duration of this project. Amongst whom I must distinguish Ted in always having the unique ability to act as a rubber duck[1] for me to bounce ideas off of.

To my personal tutor Jack Tian who has always encouraged me in my academic pursuits, and contributes immensely in shaping my intellectual curiousity throughout my time at the University of Surrey.

Last but not least to Dr. Martín Barrère Cambrun, the single largest contributor to the inspiration and creation of this project that goes above and beyond my expectations of an undergraduate supervisor. I give my thanks for the hours of help I have received, which Dr. Cambrun has generously been willing to expend to me.

[1] https://en.wikipedia.org/wiki/Rubber_duck_debugging

Contents

List of Figures

List of Tables

Glossary of Symbols

Graph Notations

$G = (V, E)$ The directed graph on which the game is played: V is the set of network hosts (nodes), E the set of directed edges (possible compromise relations).

V Vertex set of G; each element represents a host (machine) in the ICS.

E Edge set of G; each $(u, v) \in E$ means host u can compromise host v.

$S \subseteq V$ Source (entry) nodes from which an attacker may begin.

$T \subseteq V$ Target nodes the attacker aims to reach.

$\text{cost}(n)$ A value (e.g. monetary or risk score) assigned to node $n \in V$.

Attacker and Defender Action Sets

A_a Attacker's pure-strategy set: set of paths from any $s \in S$ to any $t \in T$.

A_d Defender's pure-strategy set: selections of edges on which to deploy honeypots, including "no action", subject to budget.

$N_{\mathbf{HP}}$ Defender's honeypot budget (maximum number of honeypots that can be deployed).

$\mathbf{comb}(\mathbf{set_{edges}}, n)$ The set of all n combinations of edges $\text{set}_{\text{edges}}$ (used to enumerate A_d when $N_{\text{HP}} > 1$).

Payoff/Reward Function

$R_d(a_d, a_a)$ Defender's payoff when defender picks $a_d \in A_d$ and attacker picks $a_a \in A_a$.

R_c Fixed reward for "capturing" an attacker (i.e. intercepting them on a honeypot).

R_e Penalty (negative reward) when the attacker reaches the target uncaught.

R_h Cost multiplier per deployed honeypot.

α Exponent weighting to penalize larger honeypot allocations (e.g. $\alpha > 1$ makes extra honeypots increasingly expensive).

$|a_d|$ Number of honeypots in action a_d.

$C = a_d \cap a_a$ Intersection of defender- and attacker-edge sets; non-empty iff the attack is intercepted.

LP Notation

$\mathbf{x} \in \mathbb{R}^{|A_d|}$ Defender's mixed-strategy vector over A_d (probabilities sum to 1).

$\mathbf{y} \in \mathbb{R}^{|A_a|}$ Attacker's mixed-strategy vector over A_a.

$\mathbf{x}^\top R_d(G)\,\mathbf{y}$ Expected defender utility under $(\mathbf{x}, \mathbf{y})$.

U_d Defender's value of the zero-sum game (optimal guaranteed payoff).

$\mathbf{x}_{n+1}$ Auxiliary variable appended to $\mathbf{x}$ in the LP formulation to represent U_d.

Core Attack Graph Notations

genCoreGraph$(G, s, t, \dots)$ Recursive procedure to compute the set of *core paths* from s to t.

computeAllCorePaths(G, S, T) Aggregates core paths for every $s \in S$, $t \in T$.

$P_s^t(G)$ All simple paths from s to t in G.

τ_{cg} Transformation mapping G to its core attack graph $C_{s,t}^G = (V_c, E_c)$.

(V_c, E_c) Vertex and edge sets of the core attack graph (union of all induced subpaths).

ω_c Optional weight function on edges of the core graph.

GENIND Parameters

G_1, G_2, G_3 GENIND-generated subgraphs for network, control, and sensor layers.

N_1, N_2, N_3 Number of nodes in the network, control, and sensor layers respectively

U_1, U_2 Number of clusters generated in the control and sensor layers by `Control_Generator` and `Sensor_Generator`

$\text{Type}_1, \text{Type}_2, \text{Type}_3$ Graph model types passed to `Network_Generator`, `Control_Generator`, and `Sensor_Generator` (e.g. Gabriel, mesh, star, Barabási–Albert)

clusters Number of clusters in both control and sensor layers.

lower_bound, upper_bound Min/max connections from each control cluster into the network layer.

G_c Combined graph after linking G_1, G_2, G_3 per Algorithm 2.

Abbreviations

AG	Attack Graph
APT	Advanced Persistent Threat
CAG	Core Attack Graph
CPS	Cyber-Physical System
ICS	Industrial Control System
IDS	Intrusion Detection System
LP	Linear Program
NE	Nash Equilibrium
SA	Security Administrator
ZS	Zero-Sum (Game)

Chapter 1

Introduction

Industrial control systems (ICSs) are a frequent target of malicious actors, with the number of attacks increasing as hackers become more advanced. 2024 saw nearly 1700 successful ransomware breaches, with that number invariably being higher taking into account that many of these attacks go unreported (Lemos 2025). This is undoubtably a topic of heavy concern among nation states. Given the inevitability of attacks, numerous efforts have been made in developing systems that effectively detect intrusions against computer networks. Part of this effort includes the development of Intrusion Detection Systems which are common devices that can be found on most public and private networks. However, as time goes on, more and more sophisticated attacks have appeared that can evade detection by masking as normal network traffic (Khraisat et al. 2019). This gives us two **aims** to fulfil, that form the basis of cyber-deception as a paradigm:

- To be able to consistently identify attacks, even when posing as normal traffic.
- Deter or discourage attackers from attacking in the first place, fearing detection.

1.1 The Approach to Defence

The proposal of this project is to present an effective method of defense by deception through the use of allocating honeypots via a game-theoretic approach applied to attack graphs. The subject of strategic honeypot allocation is as of now a growing field and as such, there are still many unanswered questions. Many difficult questions need to be addressed in the formulation

of the model and implementation for a project like this, e.g.;

- How can you accurately model the profiles of the attacker and defender so that this model can be used effectively?
- How can you test and reproduce these results given a lack of publicly available models of ICS networks to model them?
- Further, given that the attacker can (and will) move anywhere it wishes, how can we provide a computationally efficient model which captures that behaviour?

To fulfil the aims stated earlier and directly tackle these questions, it is imperative not just to understand the nature and motivations of an attacker, but also that of the defender. The following Section 1.2 gives a brief overview on Stuxnet as a case study of understanding how an attacker operates and how the defender must act in response. Section 1.3 explains the development of the field of cyber deception, and the need for integrating it into ICS security.

1.2 Stuxnet: Motivations for Attack and Defence

Stuxnet was the first ever virus that utilised multiple zero-day attacks and targetted ICSs which had previously been assumed to be very secure. Many experts conclude that at the minimum, the creators of Stuxnet utilised a vast swathe of resources that rival nation states (Nakashima and Warrick 2012), and has proven to be able to cause significant harm onto critical infrastructure. Such actors have since been dubbed the Advanced Persistent Threat (APT). In the advent of Stuxnet and the APT, a growing interest in researching prevention and detection of zero-day attacks has forced security administrators (SAs) to rethink approaches to security. In Langner 2024, the author identified two philosophies in industrial security;

1. **threat-centric security** - focusing on identifying and eliminating as many known threats as possible.
2. **infrastructure-centric security** - which focuses on creating robust and secure infrastructures, protected against a vast majority of conceivable attacks.

arguing that the latter of the two approaches maximises return on investment while addressing more types of attacks than the former. This project centers around this distinction of

infrastructure-centric security as a guiding philosophy in tackling security problems, and aims to address the looming threat that is the APT.

1.3 Defence by Deception

The biggest difficulty in defending against zero-day attacks is that they are definitionally unknowable. Since Stuxnet, SAs have shifted their approach to security to account for more unknown threats, but there are still many issues that ICSs face that cannot be addressed with traditional thinking. Deception as a form of defence has in recent decades gained popularity amongst some groups of computer scientists inspired by the success of the ARMOR software (Pita et al. 2009), which proved real world usage of game theory still in use at the LAX International Airport.

Cyber-deception is a subset of a larger security paradigm known as security games which aims to deceive attackers into performing actions which are counterproductive to their objective. In favour of an infrastructure-centric approach, this project leverages a cyber-deception technique known as honeypot allocation; which places faux hosts (or systems) connected to real networks for the purpose of luring attackers to attack them where administrators can monitor and study the attack's behaviour. After appropriately assessing the attack's risk to the system, the SA exercises their judgement on what action is deemed an appropriate response.

There have been numerous efforts made to apply game theory onto network defence, many of whom utilise the concept of honeypots to their model (Píbil et al. 2012; Kiekintveld, Lisỳ, and Píbil 2015; Durkota, Lisỳ, Bošanskỳ, et al. 2019; Anwar, Kamhoua, and Leslie 2020 further analysis of methodologies employed in these papers is done in Section 2.4). Inspired by these previous efforts, this project attempts to model an APT attacker infiltrating an ICS with multiple entry points and targets. This project takes particular inspiration by Nguemkam et al. 2024 in the use of core attack graphs (developed by Barrère et al. 2017) for honeypot allocation, but differs in the application and conceptualisation, as this project models network nodes as machines instead of vulnerabilities.

The attacker (the APT) is assumed to have an abundance of resources and knowledge of the network. The attacker also understands that the defender (the SA) of the network has employed a honeypot allocation model on the network and can deduce the budget the defender has in

employing these honeypots. The defender runs the game theoretic program and receives a probability distribution of where they should place each honeypot, and equally crucial is the defender also derives a number of honeypots to use (which may be equal or less than the budget). As each honeypot placed incurs a cost on the defender, each honeypot not placed has the benefit of not incurring any cost on the defender, but increases the risk of not having enough coverage of the network in defending against the attacker.

1.4 Project Structure

The structure of the rest of this document is as follows;

- Chapter 2 includes a review of relevant literature including background on Industrial Control Systems, Attack Graphs and Honeypots, and further attempts to cover as many state of the art game theory models on security games relevant to this project.
- Chapter 3 details the model and structure of the proposed game, including the specific algorithms used to implement the game using python, including an example game played on a simple attack graph.
- Chapter 4 includes details for the implementation of generating and preparing a graph for the proposed game, summarising this in Section 4.3 with an example game played on a more complex attack graph.
- Chapter 5 provides the results of experimentation by varying parameters and measuring the outcomes of the game, and give critical evaluation for what was achieved and what needs improvement.
- In Chapter 6, I discuss the model's feasibility if implemented for real use cases and provide critical evaluation for the overall conduct of the project.
- We conclude in Chapter 7 with what this project achieves, directions for future work in this project's model, and LSEP considerations.

In summary, I formulate a ICS network defense as a two-player security game between an attacker, and a defender deploying honeypots. Leveraging the use of attack graphs to formulate the action space and apply game-theoretic optimisation to determine how the defender should allocate honeypots under limited resources. The specific **objectives** to fulfill are;

- Develop a model for strategic honeypot allocation in ICS networks.
- Implement a topology generator simulating realistic ICS network structures.
- Employ an efficient method of reducing computational complexity through the use of CAGs.
- Evaluate the scalability and effectiveness of the proposed model.

The methodology devised for this project aims to provide a systematic approach to hardening ICS networks through the strategic placement of honeypots, and the quantity of honeypots to place before achieving diminishing returns. Through experimentation, results show that strategic allocation can significantly increase chances of detecting an intrusion before the attack can reach its desired target.

Chapter 2

Literature Review

This chapter outlines the relevant topics and foundational work in providing the background of this project. First, addressing the need for novel network hardening techniques in industrial control systems and the difficulties in implementing them. Reviewing attack graphs, why they are uniquely able to tackle existing security problems, and how they can be leveraged for honeypot allocation. Further, I discuss the limitations of traditional IDSs and suggest the use of honeypot-based cyber-deception as a method for network hardening.

2.1 Industrial Control Systems

ICSs are systems used to control and monitor industrial processes such as electricity grids, water distribution systems, oil refineries, nuclear power plants. Many of these industries are considered critical infrastructure. In recent decades, many processes have integrated cloud computing to streamline these processes (Bhamare et al. 2020), but many legacy systems remain in place that were built and designed for reliability, durability and ease of use. As such, outdated software and OS systems remain one of the largest security concerns for ICSs, having many vulnerabilities and being highly susceptible to targeted attacks by malware.

In 2003, an attack on the Davis-Besse nuclear power plant exploited a T1 bridge connection from a remote contractor bypassing the network firewall. The Slammer worm (also known as Sapphire) exploited a buffer overflow attack in the Microsoft SQL engine which had already been patched by then, but was not updated on the plant's network servers (Poulsen 2003). In the case of Stuxnet, the attackers utilised infected USB drives as its main payload delivery (Alladi,

Chamola, and Zeadally 2020). Which allowed them to install a rootkit on compatible machines, and search laterally through the network looking for a specific target PLC to inject code blocks into it (Knapp 2024).

ICSs being cyber-physical systems (CPSs) suffer some of the same difficulties when attempting to integrate cyber-security techniques. The heterogeneity of components that comprise the system, namely different hardware and software products used. Each component and method of integration can be a contributing factor to a CPS attack (Humayed et al. 2017). Communication between ICS components often rely on legacy protocols lacking basic security measures. MODBUS/TCP and DNP3 are common communication protocols in ICSs, MODBUS in particular transmits messages in clear text with no encryption, no system for integrity checking, and no way of authentication. DNP3 having a simple CRC integrity checker, also suffers from lack of encryption and authentication (Byres, Franz, and Miller 2004). With vulnerable protocols as above very much still in use today and heavily embedded into critical infrastructure (Malviya 2020; Gruenholz 2017), SAs need methods to track such existing vulnerabilities, along with evaluation techniques to track the security of any given network.

2.2 Attack Graphs

Attack Graphs (AGs) are a method of graphically representing the security of a system as a set of vertices and edges. The first use of attack graphs are attributed to Phillips and Swiler 1998 which at the time, compared to other models, had a strong ability to represent information in a digestible way for security professionals. AGs have since become one of the most widely popular frameworks for analyzing network security and have been used in network hardening applications. Ammann, Wijesekera, and Kaushik 2002 used a variant of AGs called logical AGs and proposed an algorithm for finding the minimal set of exploits required to compromise a system and argued that AGs contain more information than necessary for security analysts and introduced the monotonicity assumption. Which in plain terms mean; that attacks propagating through a graph by travelling through edges, do not return to previously travelled edges (Zhang, Wang, and Zio 2023). It is a well known issue of AGs that they frequently run into scalability issues, and require various representation techniques to reduce computational requirements.

Barrère et al. 2017 further introduced the concept of core attack graphs (CAGs), as a method for reducing the complexity of structures of AGs with a single entry and target point. The structure

of a CAG is such that it comprises of only core attack paths, where a core attack path is a set of edges connected through vertices such that each successive edge cannot be directly reached by an edge previous to its connecting edge. This will be formally discussed later in Section 4.2.

AGs are usually used for analyzing complex patterns and sets of system vulnerabilities such that they can be easily digestible for SAs. Various metrics have been developed for AG analysis and usually widely differ in applications. The Common Vulnerability Scoring System (CVSS) has often been used in various AG applications for probabilistic computations, such as calculating network exploitability (Noel and Jajodia 2014). The authors of Durkota, Lisỳ, Bošanskỳ, et al. 2019 formulated the use of CVSS scores into their model to calculate the probability of successful attacks. They study the problem of honeypot allocation on a set of common network topologies. Though not used in this project, CVSS scores may be able to offer a real quantitative component in calculating the path value of each attack path.

2.3 Honeypots (and a brief note on Intrusion Detection Systems)

This project studies the problem of optimal allocation of honeypots (HPs) and so the specifics of how they operate are beyond the scope of this project, this section provides an overview of what they are and the function they provide this project. Further discussed is the usage of Intrusion Detection Systems (IDSs).

HPs are devices that exist on networks as decoy systems for the purpose of luring attacks towards them and away from real systems. Since no real traffic should exist on a HP, anomalies can easily be identified the instance that they happen. They are valuable assets for cybersecurity researchers to use in identifying attack signatures for IDSs and operationally, they provide the same function and use as IDSs but instead of passive monitoring of legitimate traffic, they proactively trap attackers within themselves.

IDSs have long been used in industrial networks and similarly, hackers have long proven to be effective in bypassing them through evasion techniques. Discussed here are some methods employed by attackers as described in Khraisat et al. 2019. Fragmentation describes a technique which attackers use to break packets into smaller packets to be reassembled by the recipient, fragments may be sent over a long period of time which can further complicate an IDSs ability to detect fragmented attacks. Obfuscation is the technique of concealing an attack by making

the message difficult to understand. One specific method is to rewrite a known attack signature from hexadecimal encoding to Unicode which allows a character to have multiple symbolised formats, or even use double-encoding which exponentially increases the number of encodings possible. Encryption may even be used to conceal attacks targeting computer systems.

IDSs face difficult challenges in ICSs of which they alone may not be able to overcome, attackers with a certain level of skill will not be deterred by the existence of IDSs. HPs may similarly encounter issues of detection through probing however this still requires the attacker to interact with them, therefore still alerting the SA with the presence of the attacker. For these reasons, HPs remain an important asset to SAs in detecting attacks.

2.4 Game-Theory

By definition, deception as a practice must exercise some level of unpredictability, otherwise it can be fairly easy to deduce locations of HPs whether from the perspective of an attacker or not. Therefore a requirement for our allocation technique is to be non-deterministic such that a sufficiently advanced attacker is not able to reason the real location of a HP on a network. The literature on cyber-deception is heavily influenced by game-theory, with honeypot allocation being one of the most prominent methods utilising it.

The authors of Píbil et al. 2012 introduced the Honeypot Selection Game (HSG), which models HPs heterogenously as attackers will vary their strategy when faced with varying options of machines to attack. And defenders have the options in choosing a HP type. The nash equilibrium (NE) solution of the problem is found when both attacker and defender select their actions based on a probability distribution on their given options. Further, attackers with advanced capabilities are able to utilize probing techniques that may allow attackers to identify hosts that are actually HPs. The authors of Kiekintveld, Lisỳ, and Píbil 2015 reintroduced the HSG with host devices classified into categories of importance such that these values correlate to the gain/loss associated with a successful attack. In this version of the game, the defender must reason and decide what types of HPs to add to a network given that they may vary in perceived importance to an attacker. A reasonable assumption may be to choose a HP with the highest value. However, a sophisticated attacker may reason that this host is "too good to be true", and choose to attack the next best machine. Therefore, the optimal strategy is not as easily deduced upon closer inspection.

Many security games model the interaction between attacker and defender as a Stackelberg Game, which models the first player (usually the defender) as the leader and the second player (the attacker) as the follower. The follower observes the leader's action and selects a strategy based on their observation (Von Stackelberg 2010). It is assumed that the attacker is able to observe the defender and calculate an optimal strategy in response, which matches the expected behaviour of an APT. Intuitively, the first-mover is at a disadvantage because as the second-mover, you are able to modify your given strategy according the first-mover's strategy. However, this intuition only holds when the first-mover must commit to a pure strategy (a single deterministic action). When mixed-strategies (first mover gives a probability distribution of actions) are allowed, as proven by Von Stengel and Zamir 2010, the first-mover has no inherent disadvantage as long as he commits to this strategy.

In Durkota, Lisỳ, Kiekintveld, et al. 2016, the authors surveyed from 45 participants, who were attendees at a forensic malware seminar, and collected strategies for a proposed game model of network defense on three simple network structures. They found that the strategies collected tended to be simple and intuitive, and would be effective when defending against an attacker who was also another human. However, methodical attacks from seasoned adversaries may not have simple attack structures, game-theoretic modelling of optimal defense is highly likely to beat most human strategies.

Durkota, Lisỳ, Bošanskỳ, et al. 2019 later proposed a game theoretic approach to network defence by modelling the interaction between a strategic defender and attacker as a stackelberg game. Simply, the defender commits to a probabilistic mixed strategy and the attacker observes this before deploying an optimal pure strategy. The authors computed various zero sum game heuristics and on different network topologies evaluating these games based on relative regret. Their findings show that game-theoretic strategies benefit most when successfully determining the attacker's actions, however the settings of the reward structure and attack graph highly influence the defender's action.

Anwar, Kamhoua, and Leslie 2020 proposed a scalable framework for optimal HP allocation over an AG network, their results showed the model outperforming random allocation. And that their method of addressing exponential growth in complexity in the decomposition-based algorithm performs close to the true full NE solution. Though their method yields promising results, their experimentation was conducted on fairly small sized networks, and their turn based

structure may not be a realistic model for very large networks.

Sayed et al. 2024 created a novel framework applying cyber deception in deploying honeypots to dynamic networks where network topologies continuously change over time, and formulated HP deployment as a two-player Markov game and solved for defender-attacker equilibria via a Q-minimax algorithm using a compact state representation. Their results showed that strategic allocation through their GT framework significantly lowers attacker payoff. However, they solve a fundamentally different problem of modelling network mobility through the use of state-transitions.

Chapter 3

Game Model and Structure

Introduced in this section is the structure and formulation of the game model, it describes a game in which two players; an attacker and a defender, both simultaneously attempt to maximise their rewards through optimising their corresponding mixed strategy. The game is played on a graph with entry and target nodes, the goal of the attacker is to choose a path from entry to target without encountering a honeypot, while the goal of the defender is to place honeypots on the graph such that it is likely to intercept the attack path.

3.1 The Deception Game Model

Consider two strategies a defender may employ; (1) Placing all available HPs near the source nodes. (2) Placing all available HPs near the target nodes. In our game model, the attacker is modelled to have perfect knowledge of the game, and of the probability distribution of the set of defender actions. If we limit the set of defender actions to solely the edges surrounding the source nodes, an attacker that is not caught will have free reign to move through the network with no worry of encountering a HP. Similarly, if we limit the set of defender actions to solely the edges surrounding the target nodes, as long as an attacker does not approach the target nodes, he will also have free reign to move through the network with no worry of encountering a HP. Remembering that the focus of the project is not to protect the target hosts and ignoring all others, but to harden the network specifically in the case that attackers will attempt to move from source to target hosts. For these reasons, the structure of the game is modelled to discourage an advanced attacker from freely traversing a given network.

3.1.1 Modelling the game

Presented in Figure 3.1 is a simplified implementation of the game model on a graph $G = (V, E)$. where $V = \{s, 1, 2, 3, 4, 5, t\}$, and $E = \{(s, 1), (s, 2), (s, 3), (1, 4), (2, 4), (3, 5), (4, t), (5, t)\}$. The set of entry nodes $S = \{s\}$ and the set of target nodes is $T = \{t\}$ where $S, T \subseteq V$. Each node $\notin S$ will have a corresponding cost value assigned to it. The values chosen for this game are cost $= \{10, 10, 20, 30, 30, 100\}$.

The set of attacker actions A_a is all the possible attack paths from $\{S \rightarrow T\}$ corresponding to $\{(s, 1, 4, t), (s, 2, 4, t), (s, 3, 5, t)\}$. This can be algorithmically derived through a depth first search recursive algorithm as shown in Algorithm 1.

Algorithm 1 Recursive DFS for finding Attack Paths

```
procedure FindPaths(current, path, all_paths, target_node, depth)
    if depth = 0 then
        return
    end if
    if current = target_node then
        Append path to all_paths
        return
    end if
    for each neighbor in graph[current] do
        if neighbor not in path then
            FindPaths(neighbor, path ∪ [neighbor], all_paths, target_node, depth - 1)
        end if
    end for
end procedure
```

The set of defender actions is $A_d = \{\text{no action}, \{E\}\}$ when the number of honeypots $N_{\text{HP}} = 1$. When $N_{\text{HP}} > 1$;

$$A_d = \{\text{no action}, \{E\}, \{\textbf{comb}(\{E\}, 2)\}, ..., \{\textbf{comb}(\{E\}, N_{\text{HP}})\}\}$$

where $\textbf{comb}(\text{set}_{\text{edges}}, n)$ is a function that returns all possible combinations of n nodes that exist in $\text{set}_{\text{edges}}$. Given the defender choice to place 1 honeypot (N_{HP}), the set of defender actions

$A_d = \{\text{no action}, (s,1), (s,2), (s,3), (1,4), (2,4), (3,5), (4,t), (5,t)\}$.

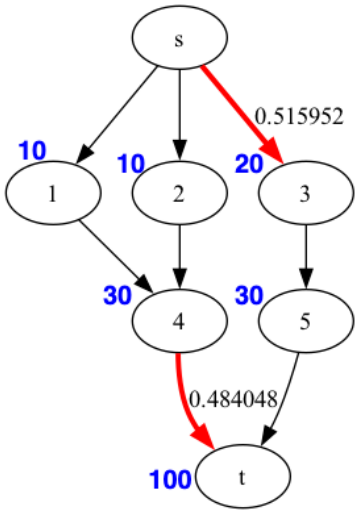


Figure 3.1: Probability distribution of placing a single honeypot on the simple game with node values (in blue).

3.1.2 The Reward Function

To identify optimal defender actions to the game model, each scenario of defender and attacker actions are considered and placed on a payoff matrix. The values in each cell of the payoff matrix correspond to the payout $R_d(a_d, a_a)$ the defender receives for his action a_d against the corresponding attacker action a_a. Given as;

$$R_d(a_d, a_a) = \begin{cases} R_c + \sum_{n \in A_a} \text{cost}(n) - (R_h \cdot \text{len}(a_d)^\alpha), & \text{if } C \neq \emptyset \text{ (attack intercepted)} \\ R_e \cdot \sum_{n \in A_a} \text{cost}(n) - (R_h \cdot \text{len}(a_d)^\alpha), & \text{if } C = \emptyset \text{ (attack succeeds)} \end{cases}$$

Where R_c is a reward for capturing the attack, R_e is the penalty for an attacker not being captured, and R_h is a honeypot cost multiplier. α exists as a weight to favour lesser honeypot allocations. $\text{len}(a_d)$ is the number of honeypots of the defender action a_d. C is the intersection of edges in the defender action and attack path:

$$C = \{a_d\} \cap \{a_a.\text{toEdges}()\}$$

These values are conceptually meant to represent real monetary impact an organisation implementing honeypots into their networks will have to take into account, however it is a loose

formulation and will likely need further adjustments to be accurate. R_c is formulated as an incentive, balancing the game structure so that when an attack is intercepted, it allows the defender to profit. Further analysis on this can be found in Section 5.2.

Table 3.1 is the Payoff Matrix for the game played in Figure 3.1. Where $R_c = 100$, $R_e = -3$, $R_h = 50$ (α is not relevant when N_{HP} is 1).

	{s,1,4,t}	{s,2,4,t}	{s,3,5,t}
No Action	-420	-420	-450
(s,1)	-90	-470	-500
(s,2)	-470	-90	-500
(s,3)	-470	-470	-100
(1,4)	-90	-470	-500
(2,4)	-470	-90	-500
(3,5)	-470	-470	-100
(4,t)	-90	-90	-500
(5,t)	-470	-470	-100

Table 3.1: Payoff matrix for a simple game.

3.1.3 Solution to the zero-sum game

The objective of the game is to obtain a probability distribution of optimal action responses $\mathbf{x}$ given the game played in G. Formulated as a zero sum game such that $R_d = -R_a$,

$$U_d(G) = \mathbf{x}^T R_d(G)\mathbf{y}$$

where $\mathbf{x}$ and $\mathbf{y}$ are mixed strategies belonging to the defender and attacker respectively.

The linear program (LP) for finding an optimal payoff U_d can be given by

$$\begin{aligned} \max_{\mathbf{x}, U_d} \quad & U_d \\ \text{subject to} \quad & \sum_{a_d \in A_d} R_d(a_d, a_a)\, \mathbf{x}_{a_d} \geq U_d, \quad \forall a_a \in A_a. \\ & \sum_{a_d \in A_d} \mathbf{x}_{a_d} = 1, \\ & \mathbf{x}_{a_d} \geq 0, \quad \forall a_d \in A_d. \end{aligned}$$

where $\mathbf{x}_{a_d}$ gives the probability of taking action $a_d \in A_d$.

An equivalent formulation that is used in this implementation appends U_d as $\mathbf{x}_{n+1}$ to $\mathbf{x}$ satisfying the following LP

$$
\begin{aligned}
\min_{\mathbf{x},\,\mathbf{x}_{n+1}} \quad & -\mathbf{x}_{n+1} \\
\text{subject to} \quad & \sum_{a_d \in A_d} R_d(a_d, a_a)\,\mathbf{x}_{a_d} - \mathbf{x}_{n+1} \geq 0, \quad \forall a_a \in A_a. \\
& \sum_{a_d \in A_d} \mathbf{x}_{a_d} = 1, \\
& \mathbf{x}_{a_d} \geq 0, \quad \forall a_d \in A_d, \\
& \mathbf{x}_{n+1} \text{is unbounded.}
\end{aligned}
$$

Used for this project is `scipy.optimize.linprog` from the SciPy (Virtanen et al. 2020) python package. The result is an array of probabilities summing to 1, with the the last element being the expected utility of the defender U_d.

The results of the simple game are $\{0, 0, 0, 0.525641, 0, 0, 0, 0.474359, 0, -289.743590\}$. Therefore, the optimal defender strategy is to place a honeypot at $(s, 3)$ with probability 0.525641 and $(4, t)$ with probability 0.474359. Which corresponds to an expected utility of -289.743590. Validating this is beyond the scope of the project (assuming the `linprog` function is correct), however a method of doing so is described in detail in Appendix A.

Chapter 4

Implementation

Listed here are deliverables implemented and presented in this chapter.

- Implement a method of random graph generation to approximate real network topologies of Industrial Control Systems as demonstrated in Section 4.1.
- Leverage efficient 'pruning' algorithms (core graph generation) to reduce the computation time (Demonstrated in Section 4.2).
- Implement a game-theoretic model (described in the previous chapter, in Section 3.1) of the honeypot allocation game on a simulated ICS topology. Such that it can determine optimal placement strategies of multiple honeypots and multiple entry and target machines. Demonstrated in Section 4.3, with evaluation of the model in Chapter 5.

4.1 Generating a Topology

There exists minimal availability of publicly accessible industrial network topologies. For financial and security reasons, organisations tend to be quite secretive about the specific configurations of their networks. Any information about the composition of ICS networks can be leveraged into attacks on them. Further, there is little doubt that there exists some ethical issues with conducting experimental attacks on real industrial topologies even for the purposes of fortifying such networks.

4.1.1 GENIND

Therefore, this project adapts a graph generator called the graph theoretic industrial topology generator (GENIND) as described in (Alrumaih and Alenazi 2023), for creating representative topologies of ICS networks. The generic structure of ICS topologies that GENIND attempts to capture through a multigraph generator are the three layers of the topology.

1. The network layer: Includes servers, domain and safety controllers, historians, firewalls. This layer primarily attempts to extract information from large data banks, and to transport data across the organization.

2. The control layer: Includes hosts such as human-machine interfaces, workstations, programmable logic controllers. This layer is responsible for optaining, processing and transmitting the dataflow from sensor layer and delivering time-sensitive services.

3. The sensor layer: Includes portable terminals, instruments, actuators, intelligent machines, smart cars, motors and pumps. Sensor devices collect parameter data and generate continuous data streams transmitted through various types of connections to the control layer in anticipation of instructions.

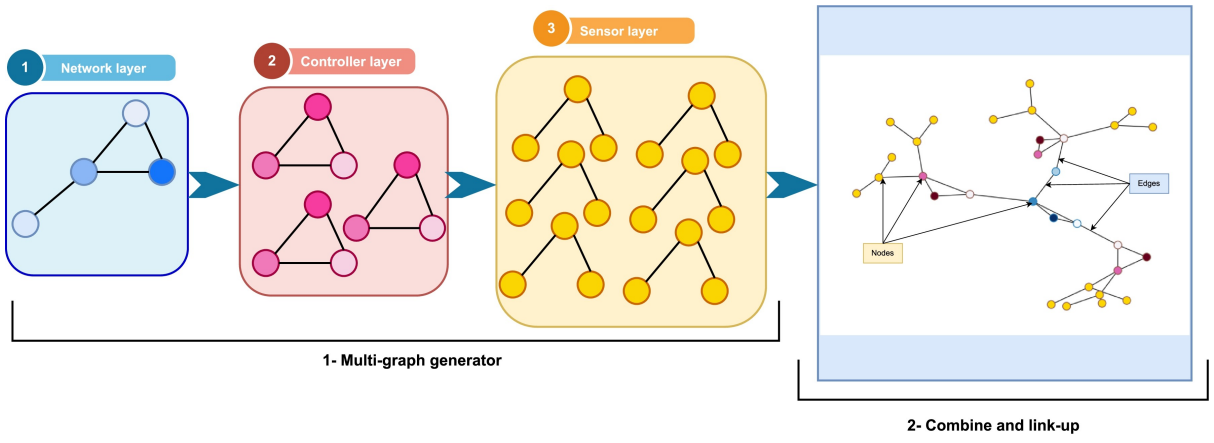


Figure 4.1: Main components of the GENIND topology generator.

The authors claim that the GENIND system closely resemble that of real network topologies due to its hierarchical structure and by creating each layer with different graph constructing techniques. The multigraph generator utilises three functions; Network_Generator(N_1, $Type_1$),

Control_Generator(N_2, $Type_2$) and Sensor_Generator(N_3, $Type_3$).

4.1.2 Modified Linkup Procedure

The algorithm used for this project linking up each layer from the multigraph generator is described in Algorithm 2. A slight modification is made in the algorithm to allow multiple connections from the network to control layer to represent more interconnected systems. First, all three separate graphs are combined into one such that they are able to form connections. Each cluster n from Control Layer G_2 selects nodes at random with a random number of connections controlled by *upper_bound* and *lower_bound*, to connect to random nodes in the network layer G_1. Then for each cluster in G_2 and G_3, a single connection is made with a random node from each cluster n.

Figure 4.2 is generated using the python library 'NetworkX' (Hagberg, Schult, and Swart 2008) showing an example of the combined graph with parameters $N_1 = 7$, $N_2 = 5$, $N_3 = 5$, $Type_1$ = barabasi-albert, $Type_2$ = ring, $Type_3$ = star, *upper_bound* = 3, *lower_bound* = 3. The figure also highlights example entry nodes N0 and N6 and target nodes S4_3 and S3_3.

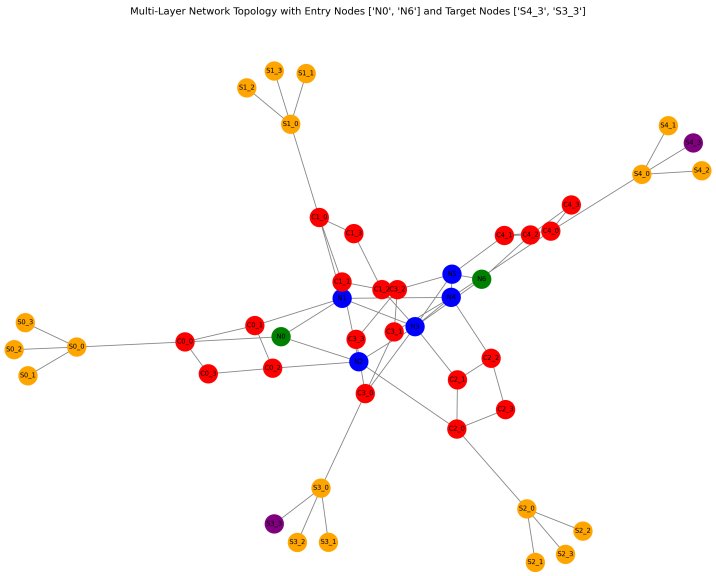


Figure 4.2: Example implementation of GENIND with parameters.

Algorithm 2 Combine And Link Up The Generated Graphs For All Zones

Require: G_1 = Network Layer, G_2 = Control Layer, G_3 = Sensor Layer

Require: *clusters* = Number of clusters for both Control and Sensor Layer

Require: *lower_bound*, *upper_bound* = Minimum and maximum number of connections from each Control cluster to the Network layer

Require: ComposeAllGraphs(G_1, G_2, G_3) - a function that combines all graphs into one

Ensure: $G_c(V_c, E_c)$ = Composed graph connected for all three layers

```
function CombineAndLinkupGraphs(G_1, G_2, G_3, clusters)
    G_c(V_c, E_c) ← ComposeAllGraphs(G_1, G_2, G_3)
    for all n ∈ clusters do
        connections ← Random integer from lower_bound to upper_bound
        for i = 1 to connections do
            G_c.AddEdge(Random(G_2[n]), Random(G_1))
        end for
    end for
    for all n ∈ clusters do
        if G_2 and G_3 are not empty then
            G_c.AddEdge(Random(G_2[n]), Random(G_3[n]))
        end if
    end for
    return G_c(V_c, E_c)
end function
```

4.2 Creating Attack Graphs From Network Topologies

Usual approaches to AGs attempt to capture the possible paths an attacker may take to achieve a target goal (i.e.; gain root access), and illustrating the dependencies between vulnerabilities in the system. With the network topology of the ICS, many host systems can have bidirectional connections. The generated topology is modelled as all edges being bidirectional such that any host may compromise another host. This is of course an unrealistic assumption, and for generating attack paths, will introduce many unnecessary calculations. To solve this, (Nguemkam et al. 2024) utilised the concept of 'Core Attack Graphs' (CAGs) formulated by (Barrère et al. 2017) as a method for reducing the number of paths to be calculated. Incidentally, by changing the

network topology from undirected to an acyclic directed subgraph of the original, the unrealistic assumption of any host being able to compromise another is mitigated. The graph is reduced such that any host that escalates the attacker's position in a network toward a target host is vulnerable to an attack. Though not perfectly realistic, it is sufficient in modelling the complex structure of attack paths that can exist on the topology.

4.2.1 Core Attack Graphs

CAGs are formulated to summarise multiple alternative vectors of attack between any two nodes in the input graph, such that the only paths that can be obtained from the CAG cannot be summarised into any other graph link. These are called core paths and share the same properties of what is known in graph theory as induced paths (Nešetřil and Mendez 2012), formally defined in Definition 1. A CAG therefore is essentially the set of all nodes and vertices that exist in all core paths from s to t. Formally defined in Definition 2. (These definitions are taken from Barrère et al. 2017)

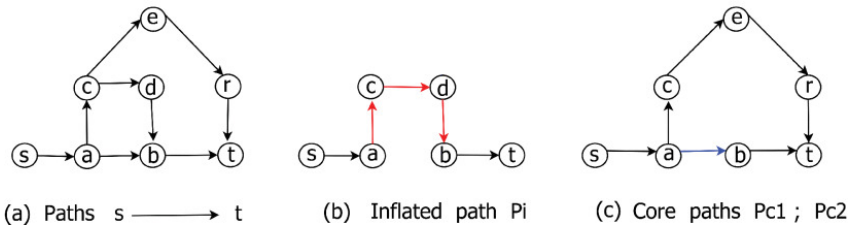


Figure 4.3: (a) Superimposed paths, (b) inflated path, (c) core graph.

Definition 1 (Core Path). *A path $p(v_0, \ldots, v_n)$ in $G = (V, E)$, $n > 0$, is a core path if and only if there is no other path $p' \neq p$ such that $p' \preceq p$, i.e.:*

$$\forall v_i \in p, \forall k \in [2, n-i], (v_i, v_{i+k}) \notin E \tag{4.1}$$

Definition 2 (Core Attack Graph). *Given a digraph $G = (V, E) \in \mathcal{G}$, a source node $s \in V$, and a target node $t \in V$, the corresponding core graph $C^G_{s,t} = (V_c \subseteq V, E_c \subseteq E, \omega_c)$ is the result of a transformation $\tau_{cg}(G, s, t)$ defined as the union of core paths from s to t in G as follows:*

$$\tau_{cg}(G, s, t) \equiv C^G_{s,t} \equiv \bigcup_{p \in P^G_{s,t}} p \ s.t. \ \nexists p' \in P^G_{s,t}, p' \preceq p \tag{4.2}$$

For the purpose of this project, the CAG generation algorithm is adapted from (Barrère et al. 2017) to model graph edges E as the ability of one host $v \in V$ being able to compromise another host $v' \in V$. This differs from the original implementation which models edges as the dependencies of vulnerabilities. The generation algorithm is described below in Algorithm 3.

Applying this generation technique to the network topology generated in Figure 4.2, produces the CAG illustrated in Figure 4.4.

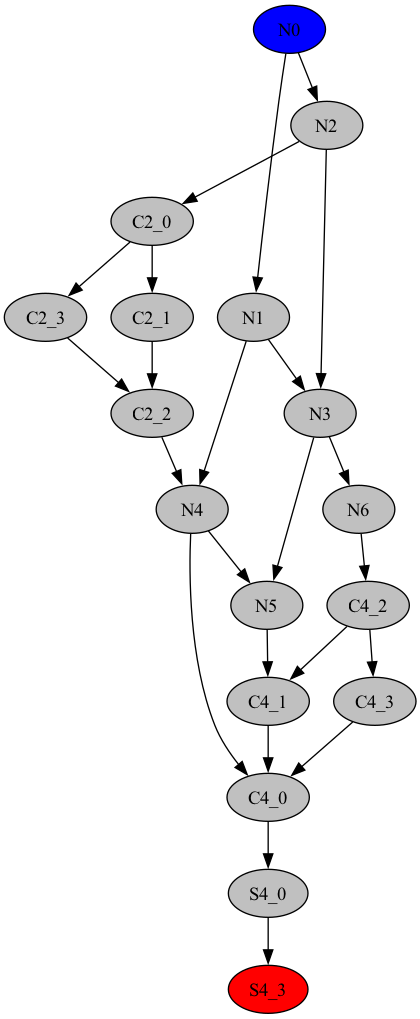


Figure 4.4: Core attack graph generated from Figure 4.2.

4.2.2 CAGs For Multiple Entry And Target Nodes

To realistically model an attacker's behaviour within a system, multiple points of entry and targets must be considered. One problem that arises with this is the introduction of cycles in

Algorithm 3 Generate Core Attack Graph

Require: Graph G, start node s, target node t, path path, set L, core paths core_paths

Require: IS_INDUCED(path, node, G) - a function that checks if adding 'node' to the current path keeps the path induced.

```
1: function genCoreGraph(G, s, t, path, L, core_paths)
2:     newpath ← path + [s]
3:     V_C ← ∅
4:     for all c ∈ core_paths do
5:         add c to V_C
6:     end for
7:     if s ∈ V_C then
8:         add s to V_C
9:         return //stop
10:    end if
11:    neighbours ← G[s] or ∅ if s = ∅
12:    if t ∈ neighbours and is_induced(newpath, L, G) then
13:        add newpath + [t] to core_paths
14:        return //stop
15:    else
16:        L_new ← L + neighbours + {s}
17:        for all n ∈ neighbours do
18:            if n ∉ L then
19:                if is_induced(newpath, n, G) then
20:                    genCoreGraph(G, n, t, newpath, L_new, core_paths)
21:                end if
22:            end if
23:        end for
24:    end if
25: end function
```

the created graph. However, this remains a non-issue as the attack paths can be computed for each entry and target combination before combining them together. Algorithm 4 describes this procedure.

Algorithm 4 Compute All Core Attack Paths

Require: Graph G, set of entry nodes $entry\_nodes$, set of target nodes $target\_nodes$

Ensure: All core paths between entry and target nodes

```
1:  function computeAllCorePaths(G, entry_nodes, target_nodes)
2:      all_core_paths ← []
3:      for all s ∈ entry_nodes do
4:          for all t ∈ target_nodes do
5:              core_paths ← []
6:              genCoreGraph(G, s, t, [], ∅, core_paths)
7:              append core_paths to all_core_paths
8:          end for
9:      end for
10:     return all_core_paths
11: end function
```

Applying Algorithm 4 to the ICS topology in Figure 4.2 produces the combined attack graph shown in Figure 4.6(a).

4.3 The Game Action Space as Generated by the Framework So Far

Summarising the overall implementation, we start with defining the parameters for the topology we are trying to simulate and create a synthetic ICS topology through the GENIND generator. These parameters are; the **number of network nodes**, the **number of clusters**, and the **number of nodes per cluster**.[1] After creating the topology, the graph is composed and reformatted such that further processing can be done. The CAGs are computed for each entry

[1]There is a fourth parameter worked in to fix the number of connections from the Control to Sensor layer so that we can deterministically recreate the graphs for evaluation.

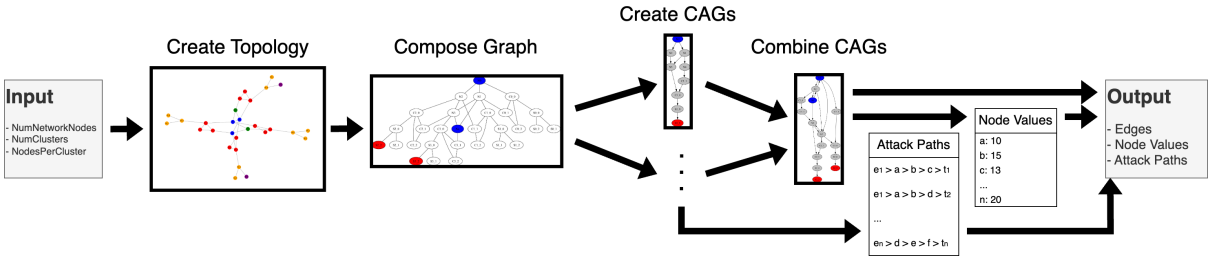


Figure 4.5: Pipeline for building the combined CAG for use as the game action space.

and target pair, and are combined into a full CAG. With this, random values are assigned to each node in the graph,[2] and the output of the pipeline is; the **set of edges of the full CAG**, the **list of node values**, and the **list of attack paths**.

With this implementation, the goal is to now play the deception game on the action space as produced by the pipeline as described above (and presented in Figure 4.5).

4.3.1 The game played on the combined CAG

To recapitulate the game model in Chapter 3, the defender's action $a_d \in A_d$, constrained by the budget N_{HP} such that $\text{len}(a_d) \leq N_{\text{HP}}$, is pitted against the attacker's action $a_a \in A_a$ which is a set of nodes $\in V$ which span the attack graph $G(V, E)$ from one entry node to a target node.

The payoff table is a matrix of payoff values given by $R_d(a_d, a_a)$, where A_d comprises the set of row actions and A_a comprises the set of column actions. The solution of which is a probability distribution of defender actions which maximise the defender's payoff, calculated by a linear programming function.

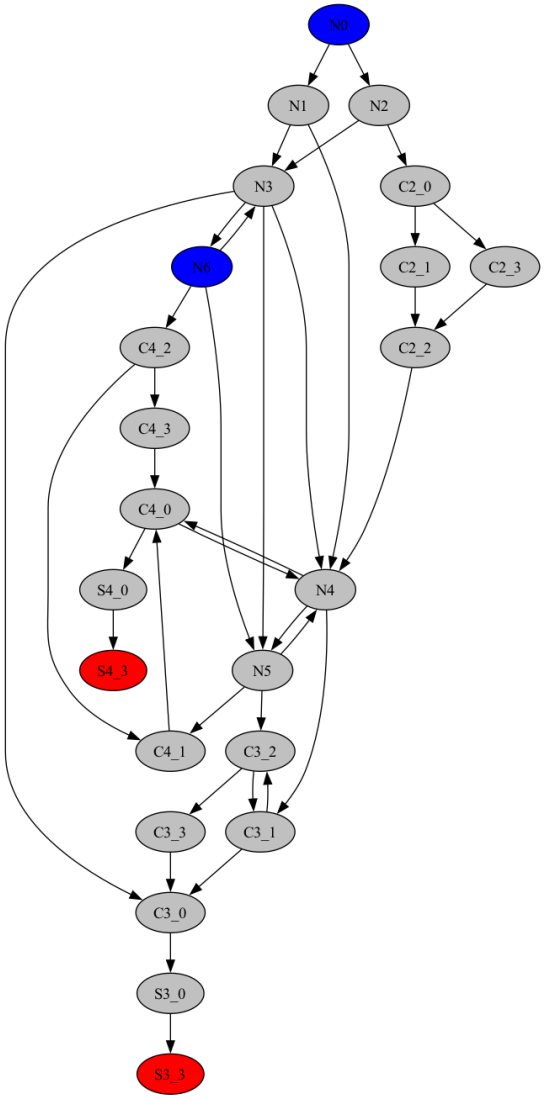
Figure 4.6(b) shows output of the program highlighting the edges which comprise the mixed strategy profile of the defender action based on a weighing function that increases the thickness of the edge according to the probability distribution shown in Table 4.1, which shows the top 20 actions the defender should take according to the probability of choosing that action.

An important modification to the generation of defender actions A_d that needs to be mentioned is that for the game to produce more diverse strategy profiles consistently, the set of defender actions was modified to disallow placement on edges connecting to sensor nodes.

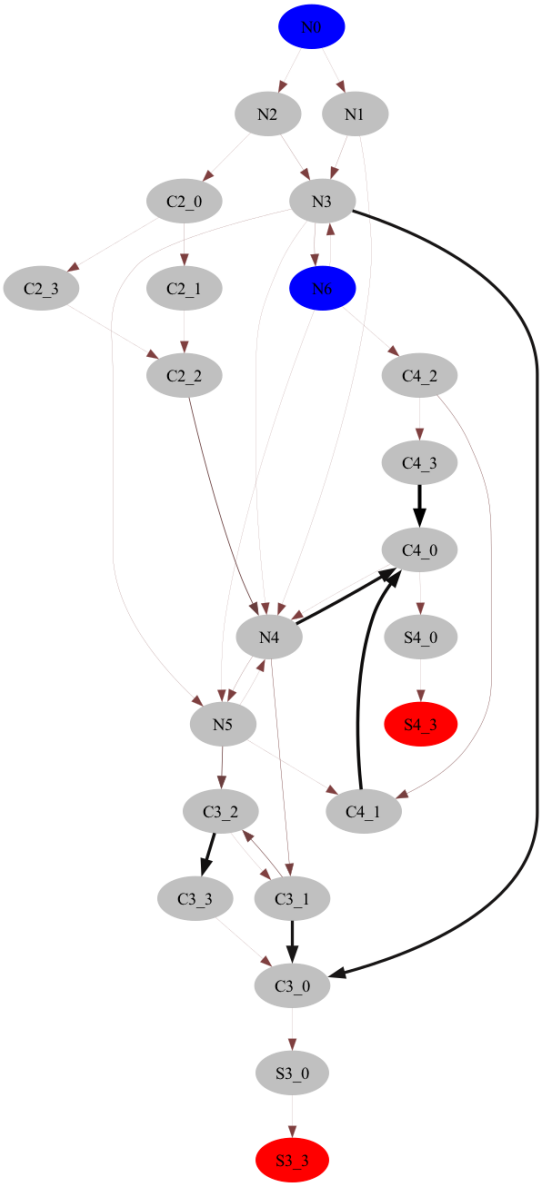
[2] These values are deterministically set by a seed in evaluation.

Probability	**Defender Action (Set of Edges)**
0.3140	{('C3_1', 'C3_0'), ('C4_1', 'C4_0'), ('N4', 'C4_0')}
0.1782	{('C3_2', 'C3_3'), ('C4_3', 'C4_0'), ('N3', 'C3_0')}
0.1348	{('C3_2', 'C3_3'), ('C4_1', 'C4_0'), ('C4_3', 'C4_0')}
0.0948	{('C3_1', 'C3_0'), ('C3_2', 'C3_3'), ('N3', 'C3_0')}
0.0612	{('C2_2', 'N4'), ('C4_3', 'C4_0'), ('N3', 'C3_0')}
0.0578	{('C4_3', 'C4_0'), ('N4', 'C4_0'), ('N5', 'C3_2')}
0.0542	{('C3_1', 'C3_2'), ('C4_3', 'C4_0'), ('N4', 'C4_0')}
0.0250	{('C4_2', 'C4_1'), ('N3', 'C3_0'), ('N4', 'C3_1')}
0.0237	{('C3_2', 'C3_3'), ('N3', 'C3_0'), ('N5', 'C3_2')}
0.0128	{('C2_2', 'N4'), ('N3', 'C3_0'), ('N3', 'N6')}
0.0103	{('C4_3', 'C4_0'), ('N4', 'C3_1'), ('N4', 'N5')}
0.0100	{('C2_2', 'N4'), ('N1', 'N3'), ('N2', 'N3')}
0.0073	{('C2_0', 'C2_1'), ('C4_2', 'C4_1'), ('N3', 'C3_0')}
0.0053	{('C2_2', 'N4'), ('C4_3', 'C4_0'), ('N3', 'N5')}
0.0037	{('C3_1', 'C3_2'), ('N3', 'N6'), ('N4', 'C4_0')}
0.0027	{('C2_0', 'C2_1'), ('C4_3', 'C4_0'), ('N3', 'C3_0')}
0.0020	{('N3', 'C3_0'), ('N4', 'C3_1'), ('N4', 'N5')}
0.0012	{('C2_2', 'N4'), ('C4_3', 'C4_0'), ('N5', 'N4')}
0.0005	{('C2_2', 'N4'), ('N1', 'N4'), ('N2', 'N3')}
0.0005	{('C2_2', 'N4'), ('C4_1', 'C4_0'), ('N1', 'N4')}

Table 4.1: Probabilities of choosing corresponding sets of edges



(a) Full CAG produced for two entry and two target nodes on the topography generated in Figure 4.2.



(b) Sample game played on the CAG produced in Figure 4.6(a) with 3 honeypots.(Highlighted lines are weighted based on their probabilities as shown in Table 4.1)

Figure 4.6: Full CAG and Game Result produced from the GENIND topology with two entry nodes (N0, N6) and two target nodes (S3_3, S4_3).

Chapter 5

Results

5.1 Evaluation Setup

Network	nn	nc	npcl	fc
1 (**Mini**)	4	2	2	1
2 (**Small**)	5	3	2	2
3 (**Medium**)	6	4	3	3
4 (**Large**)	7	5	4	3

Table 5.1: Network Parameters for Different Configurations; **nn** (number of network nodes), **nc** (number of clusters), **npcl** (number of nodes per cluster), **fc** (fixed number connections)

A series of experiments are used to conduct evaluation on the performance and characteristics of this model on different ICS graphs. For these experiments, different sized graphs with varying numbers of connections between the network and controller layer are considered, this is implemented via a function which builds the game action space as described in Section 4.3 and visualised in Figure 4.5. Table 5.1 gives the 4 compositions and their parameters considered in producing the evaluation graphs shown in Figure 5.1. Admittedly, the values for each configuration are arbitrarily chosen, however they are meant to show meaningful differences in values when calculating results, as varying each one manually will not yield intelligible graphs. To produce Figure 5.2, we only consider a honeypot budget N_{HP} of 3. Only two entry and target nodes are considered for all network configurations and are deterministically set according to

the following parameters;

$$\text{entry\_nodes} = [\text{N}_{[\text{first node}]}, \text{N}_{[\text{last node}]}],$$

$$\text{target\_nodes} = [\text{S}_{[\text{last node in last cluster}]}, \text{S}_{[\text{last node in second last cluster}]}]$$

All experiments use the following parameters $R_c = 2000$, $R_e = -1$, $R_h = 200$ and $\alpha = 1.125$, in setting up the game.

5.2 Experimental Analysis

Evaluating the performance of this honeypot allocation model involves calculating its effectiveness and scalability measured by varying network sizes and the honeypot budget. From here, optimal payoff and optimal value are used interchangeably for the measure of U_d.

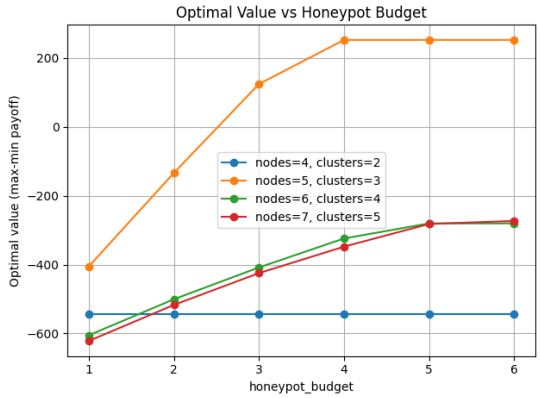
In Figure 5.1(a), we see the effect of increasing the honeypot budget N_{HP} on the optimal payoff U_d calculated by the linear program. There is no pattern of increase according to the network sizes but it shows that there is an 'optimal budget' of honeypots where a higher number no longer helps to increase the payoff.

Figure 5.1(b) shows that generally the processing time increases exponentially according to N_{HP}, with the exception when the budget is 1, which is likely because of overhead calculations made by the linear program.

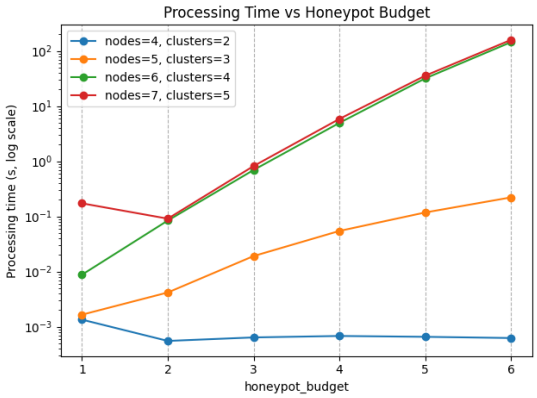
Figure 5.1(c) shows the defender action space A_d growing exponentially with N_{HP} increasing, this is expected as A_d is a combinatorial number of honeypots based on the number of allowable edges. Network 1 (Mini) is unseen as it is overshadowed by Network 2 (Small).

Based on the findings in exponential scaling with processing time and action space A_d according to the honeypot budget, these two values are likely correlated. And we see in Figure 5.1(d) that the combined action (A_d, A_a) space scales linearly to processing time. Networks 1 and 2 (Mini and Small) have very small action spaces and minimal processing time and as such; they do not appear.

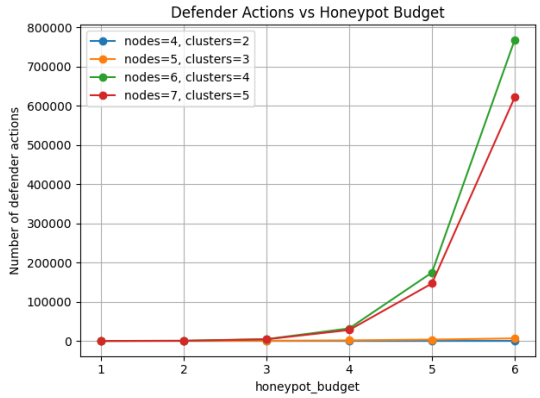
Figure 5.2(a) shows that optimal payoff U_d generally decreases as honeypot cost R_h increases. This is likely influenced by the honeypot scaling factor α which favours lesser honeypot allocations. When R_h increases, the model starts to favour defender actions a_d with less allocations which will increase U_d.



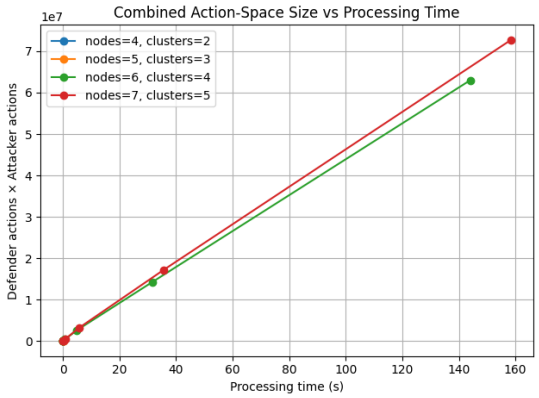
(a) Optimal value increases until it hits a 'ceiling' and then cannot increase due to already reaching optimal number of honeypots.



(b) Processing time spikes by orders of magnitude with higher honeypot budgets for larger configurations.



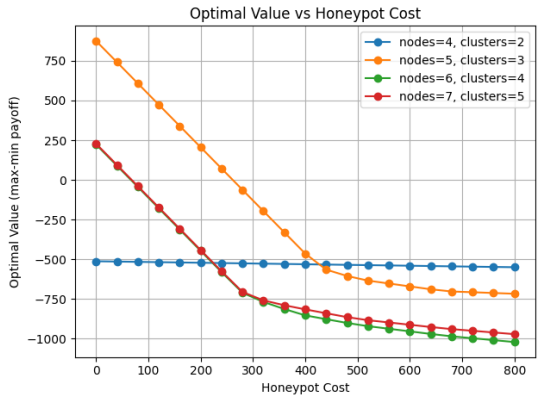
(c) Defender actions grows exponentially with honeypot budget, larger setups exhibit dramatically higher action counts.



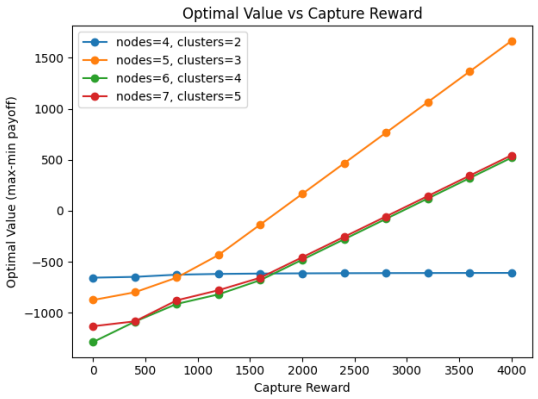
(d) The total number of defender × attacker actions scales linearly with processing time across all node and cluster configurations.

Figure 5.1: Comparison of various evaluation results varying each game by the honeypot budget

Figure 5.2(b) shows almost linear scaling for varying capture reward, however degenerates when the capture reward is close to 0. This can be explained by a couple of factors; when the reward for capture is so small, the model may not be incentivised to act at all if the cost of allocation exceeds the reward, therefore resulting in a more nuanced defender strategy where it may consider no allocation. Because the decision threshold for whether an attack should be caught or not may be very small, the results can be influenced by small variations in any of the following; node values, capture reward, scaling factor, budget, etc.



(a) Varying the honeypot cost on the optimal defender payoff U_d shows diminishing returns for higher deployment costs.



(b) Varying the capture reward R_c on optimal payoff U_d shows almost linear scaling when R_c is sufficiently high.

Figure 5.2: Measure of optimal value U_d when varying honeypot cost R_h and capture reward R_c, with $N_{\text{HP}} = 3$.

Chapter 6

Discussion

Many existing game-theory models require an advanced degree of understanding in game theory, cyber security, and in some cases, knowledge of attack graphs. I believe the formulation of this model is an approachable framework that does not require a user to have much advanced knowledge to utilise as a tool, though assigning node values may require further analysis on the user's side. The user must understand that the output is a probability distribution of actions to employ and decide what to use it for, though that is beyond the scope of this framework, they may choose to run a randomiser to select which action to employ based on its probability, possibly employ a rank based selection of the top n values. They may even decide to not use this framework for real application but strictly to analyse the security of their ICS network.

A novel technique this project proposes is the concept of a full CAG (Section 4.2.2) which builds off of Core Attack Graphs in Barrère et al. 2017. Nguemkam et al. 2024 used CAGs to efficiently approximate the optimal game value but only utilised the concept for a single entry and target node. My proposal was to combine the CAGs of each entry node to each target node (calculating the core paths each time, thus avoiding the problem of cycles being introduced) and use the combined full CAG for approximating the optimal value. Though not a particularly complex idea, to the best of my knowledge, it is still a novel formulation in generating attack graphs for multiple points of entries and targets on the same graph.

6.1 Critical Evaluation

It is difficult to say whether this model has much real world application, certainly in its current form it has many areas to improve in. Firstly, the reward function is somewhat arbitrary in its formulation and can certainly benefit from more research, one problem with it is the non-capture penalty R_e. I initially reasoned that a multiplier effect for penalizing successful attacks could be useful to weigh them more negatively. However, a larger R_e proved difficult to optimize for, and most of my experimentation could only resolve when the value was closer to 1.

Next, the proposed attacker model which is able to attack any and all machines is unreasonable, this is currently ignored because the game is modelled on a CAG of a random graph. Therefore, it is assumed that paths that the attacker cannot take are already ignored by playing the game on the CAG. However, if the game is played on a real ICS topology, some further formulation of which nodes can and cannot be attacked are required to produce a reasonable CAG, possibly through the use of external Attack Graph tools such as MulVal (Ou, Govindavajhala, Appel, et al. 2005).

Another issue is that with the way GENIND (Section 4.1) creates graphs, the defender action space needs to be reduced to exclude sensor nodes because the model will usually favour placing honeypots on those edges, as it is usually the most efficient strategy. More research is required in determining whether this topology formulation can accurately simulate attacks and whether a more realistic defender action space is needed.

This model works at a fairly efficient pace for minimal honeypot allocations but starts requiring large processing times for larger networks and larger honeypot budgets. This is largely due to the set of defender actions exploding in size as the number of combinations of placements scales exponentially. Some preparation in determining whether a large honeypot budget is even needed may be used, as evidenced by the 'ceiling' shown in Figure 5.1(a) and estimating the processing time can be done by extrapolating Figure 5.1(d).

In summation; the model has difficulty scaling per the honeypot budget N_{HP} and with larger network sizes. Further, there are issues with balancing realistic parameters with producing compelling results for the topology. And in general, it is difficult to evaluate whether this specific model is usable for real world application.

Chapter 7

Conclusion

This project has explored and demonstrated a working game theoretic model which leverages a cyberdeception technique known as honeypot allocation for hardening network security in industrial control systems. As opposed to other works, this project focuses on the application to ICS networks with a simplified model that can consider multiple entry and target nodes. Implementation of this model is done by applying it to an action space generated by a novel topology generator called GENIND and 'pruned' through producing CAGs for each entry and target pair. Several experiments are conducted on various topology sizes to evaluate the scalability and performance of the proposed model, with further analysis on the effects of varying the capture reward, and the budget and cost of honeypots. This model's application in the real world and its limitations are discussed, with critical evaluation on how this project was generally handled.

The following is a brief summary of what this project has achieved;

- Evaluate the state of the art in game theory techniques for honeypot allocation.
- Provide a working model for application specifically for ICS networks.
- Proposed and implemented a suitable framework for applying the model onto generated ICS topologies, including creating and combining CAGs of the topology.
- Evaluated the performance of the model in terms of scalability, and
- Analysed the sensitivity and effects of changing various parameters on the game's outcome.

Successfully fulfilling the aims and objectives laid out in Chapter 1.

7.1 Future Work

Future work in evaluating the effect of using the full attack graph instead of CAGs may be useful, and would validate if Nguemkam et al. 2024's results of negligible difference between the two hold true for our formulation of the game. Further, there may be room for further efficient computation by utilising or formulating a 'pruning' algorithm which can further reduce the defender and attacker action space.

There are likely solutions to the issues detailed in Section 6.1 but not implemented due to time constraints. For example, a turn-based game in which an attacker would move through the system and the defender deciding whether and where it should allocate a honeypot based on the attacker's position (such as the one proposed in Anwar, Kamhoua, and Leslie 2020) may solve the issue of exponential growth in action-space. However, this requires a drastic reformulation of the game.

However, there is likely future potential for a more efficient method of generating a reduced defender action set such that unlikely choices are ignored, or possibly combine similar actions such that they summarise actions in a similar way that CAGs do.

More experimentation with the payoff function in Section 3.1.2 can be conducted for more realistic results, however an objective metric comparison should then be considered. One possible comparison could be to conduct a random allocation strategy. However, a randomized strategy will not yield stable results and may not be a valid method of evaluation.

7.2 Legal, Social, Ethical and Professional Issues (LSEP)

The subject of honeypots is of important concern in the security community and to this paper. The application of HPs and whether they are considered enticement or entrapment, and where that line can be drawn is hazy, not very well understood, **but is the difference between what is Legal and Illegal**. Administrators who opt to use honeypots must have a high level of expertise in their field and should consult legal professionals in determining their usage. The framework developed in this project, in and of itself would likely not violate any laws. However, there is always the chance that some administrator with or without the understanding of legal issues with honeypots, may utilise this framework for the purpose of entrapment.

The **Social Responsibility** of taking up this project includes running the risk of the above, however I believe that an effective method of implementing game theory for security purposes can, and has proven to (Pita et al. 2009) have greater societal impacts.

As far as I understand, under consultation with this project's supervisor Dr. Martín Barrère Cambrun, this project **does not involve any Ethical Issues**, uses no external datasets, and does not involve any data collection, and has been approved for submission.

In the British Computer Society (BCS) Code of Conduct[1], they lay out some considerations for professional competence and integrity, I take these as reference for my **Professional Conduct** and profess that I adhere to all points listed below;

1. I have only undertaken work that is within my professional competence, and do not claim any level of competence that I do not posess.

2. I strive to develop my professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards relevant to this field.

3. I ensure I have sufficient knowledge and understanding of legislation and comply with such legislation in carrying out my professional responsibilities.

4. I respect and value alternative viewpoints and have sought out and taken in honest criticisms of my work, particularly in conversations and scheduled meetings with my supervisor Dr. Barrère.

5. I have not been made any offer of bribery or unethical inducement.

[1] https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/

Bibliography

Alladi, Tejasvi, Vinay Chamola, and Sherali Zeadally (2020). "Industrial control systems: Cyberattack trends and countermeasures". In: *Computer Communications* 155, pp. 1–8.

Alrumaih, Thuraya N.I. and Mohammed J.F. Alenazi (2023). "GENIND: An industrial network topology generator". In: *Alexandria Engineering Journal* 79, pp. 56–71. ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2023.07.062. URL: https://www.sciencedirect.com/science/article/pii/S111001682300649X.

Ammann, Paul, Duminda Wijesekera, and Saket Kaushik (2002). "Scalable, graph-based network vulnerability analysis". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 217–224.

Anwar, Ahmed H, Charles Kamhoua, and Nandi Leslie (2020). "Honeypot allocation over attack graphs in cyber deception games". In: *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, pp. 502–506.

Barrère, Martín et al. (2017). "Tracking the bad guys: An efficient forensic methodology to trace multi-step attacks using core attack graphs". In: *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–7. DOI: 10.23919/CNSM.2017.8256038.

Bhamare, Deval et al. (2020). "Cybersecurity for industrial control systems: A survey". In: *computers & security* 89, p. 101677.

Byres, Eric J, Matthew Franz, and Darrin Miller (2004). "The use of attack trees in assessing vulnerabilities in SCADA systems". In: *Proceedings of the international infrastructure survivability workshop*. Vol. 202. Citeseer.

Durkota, Karel, Viliam Lisỳ, Branislav Bošanskỳ, et al. (2019). "Hardening networks against strategic attackers using attack graph games". In: *Computers & Security* 87, p. 101578.

Durkota, Karel, Viliam Lisỳ, Christopher Kiekintveld, et al. (2016). "Case studies of network defense with attack graph games". In: *IEEE Intelligent Systems* 31.5, pp. 24–30.

Gruenholz, Julie (Sept. 2017). *Why, in 2017,are we still using Modbus?* URL: https://www.hallam-ics.com/blog/why-in-2017-are-we-still-using-modbus.

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart (2008). "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, pp. 11–15.

Humayed, Abdulmalik et al. (2017). "Cyber-physical systems security—A survey". In: *IEEE Internet of Things Journal* 4.6, pp. 1802–1831.

Khraisat, Ansam et al. (2019). "Survey of intrusion detection systems: techniques, datasets and challenges". In: *Cybersecurity* 2.1, pp. 1–22.

Kiekintveld, Christopher, Viliam Lisỳ, and Radek Píbil (2015). "Game-theoretic foundations for the strategic use of honeypots in network security". In: *Cyber Warfare: Building the Scientific Foundation*, pp. 81–101.

Knapp, Eric D. (2024). "7 - Hacking Industrial Control Systems". In: *Industrial Network Security (Third Edition)*. Ed. by Eric D. Knapp. Third Edition. Syngress, pp. 181–229. ISBN: 978-0-443-13737-2. DOI: https://doi.org/10.1016/B978-0-443-13737-2.00001-4.

Langner, Ralph (Sept. 2024). *Threat-centric vs. infrastructure-centric OT security*. URL: https://www.langner.com/2024/04/threat-centric-vs-infrastructure-centric-ot-security/.

Lemos, Robert (2025). *Industrial System Cyberattacks Surge as OT Stays Vulnerable*. URL: https://www.darkreading.com/cyber-risk/industrial-system-cyberattacks-surge-ot-vulnerable.

Malviya, Nitesh (Mar. 2020). *Modbus, DNP3 and Hart*. URL: https://www.infosecinstitute.com/resources/scada-ics-security/modbus-dnp3-and-hart/.

Nakashima, Ellen and Joby Warrick (June 2012). *Stuxnet was work of U.S. and Israeli experts, officials say*. URL: https://www.washingtonpost.com/world/national-security/stuxnet-was-work-of-us-and-israeli-experts-officials-say/2012/06/01/gJQAlnEy6U_story.html.

Nešetřil, Jaroslav and Patrice Ossona de Mendez (2012). "Bounded Height Trees and Tree-Depth". In: *Sparsity: Graphs, Structures, and Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 115–144. ISBN: 978-3-642-27875-4. DOI: 10.1007/978-3-642-27875-4_6. URL: https://doi.org/10.1007/978-3-642-27875-4_6.

Nguemkam, Achile Leonel et al. (2024). "Optimal Honeypot Allocation Using Core Attack Graph in Partially Observable Stochastic Games". In: *IEEE Access* 12, pp. 187444–187455. DOI: 10.1109/ACCESS.2024.3513461.

Noel, Steven and Sushil Jajodia (2014). "Metrics suite for network attack graph analytics". In: *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. CISR '14. Oak Ridge, Tennessee, USA: Association for Computing Machinery, pp. 5–8. ISBN: 9781450328128. DOI: 10.1145/2602087.2602117. URL: https://doi.org/10.1145/2602087.2602117.

Ou, Xinming, Sudhakar Govindavajhala, Andrew W Appel, et al. (2005). "MulVAL: A logic-based network security analyzer." In: *USENIX security symposium*. Vol. 8. Baltimore, MD, pp. 113–128.

Phillips, Cynthia and Laura Painton Swiler (1998). "A graph-based system for network-vulnerability analysis". In: *Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79.

Píbil, Radek et al. (2012). "Game theoretic model of strategic honeypot selection in computer networks". In: *Decision and Game Theory for Security: Third International Conference, GameSec 2012, Budapest, Hungary, November 5-6, 2012. Proceedings 3*. Springer, pp. 201–220.

Pita, James et al. (2009). "Armor software: A game theoretic approach to airport security". In: *Protecting Airline Passengers in the Age of Terrorism (2009)* 163.

Poulsen, Kevin (Aug. 2003). *Slammer worm crashed Ohio Nuke Plant Net*. URL: https://www.theregister.com/2003/08/20/slammer_worm_crashed_ohio_nuke/.

Sayed, Md Abu et al. (2024). "Strategic Honeypot Allocation in Dynamic Networks: A Game-Theoretic Approach for Enhanced Cybersecurity". In: *Research Square*. DOI: 10.21203/rs.3.rs-3960163/v1. URL: https://doi.org/10.21203/rs.3.rs-3960163/v1.

Virtanen, Pauli et al. (Mar. 1, 2020). "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3, pp. 261–272. DOI: 10.1038/s41592-019-0686-2. URL: https://doi.org/10.1038/s41592-019-0686-2.

Von Stackelberg, Heinrich (2010). *Market structure and equilibrium*. Springer Science & Business Media.

Von Stengel, Bernhard and Shmuel Zamir (2010). "Leadership games with convex strategy sets". In: *Games and Economic Behavior* 69.2, pp. 446–457.

Zhang, Jinghan, Wei Wang, and Enrico Zio (2023). “Study on the Application of Graph Theory Algorithms and Attack Graphs in Cybersecurity Assessment”. In: *2023 7th International Conference on System Reliability and Safety (ICSRS)*. IEEE, pp. 558–564.

Appendix A

Validating Optimal Payoff

The calculation of the optimal value is given by the result of the LP implemented in SciPy's `scipy.optimize.linprog` as discussed in 3.1.3. This is trusted to be implemented correctly, however for peace of mind, we can further validate these results by additional computation of the attacker's LP and computing the results of both players strategies externally. Note; the original LP which returns the defender's mixed strategy also computes the attacker's mixed strategy, but does not return it.

A.1 Attacker's dual linear program

Given the zero-sum formulation of the game, the attacker's reward matrix is defined as the negative of the defender's reward matrix, i.e., $R_a = -R_d$.

The attacker's goal is to maximise their expected utility U_a under the constraint that the defender will act optimally. This results in the following linear program:

$$\begin{aligned} \max_{\mathbf{y}, U_a} \quad & U_a \\ \text{subject to} \quad & \sum_{a_a \in A_a} -R_d(a_d, a_a)\, \mathbf{y}_{a_a} \leq U_a, \quad \forall a_d \in A_d \\ & \sum_{a_a \in A_a} \mathbf{y}_{a_a} = 1 \\ & \mathbf{y}_{a_a} \geq 0, \quad \forall a_a \in A_a \end{aligned}$$

Solving this linear program returns the optimal mixed strategy for the attacker, where $\mathbf{y}_{a_a}$

represents the probability of selecting action $a_a \in A_a$.

A.2 Calculating the Game Result

Consider a zero-sum game with payoff matrix $P \in \mathbb{R}^{m \times n}$. Let

$$\mathbf{x}^* = (\mathbf{x}_1^*, \ldots, \mathbf{x}_m^*)^\top, \quad \mathbf{y}^* = (\mathbf{y}_1^*, \ldots, \mathbf{y}_n^*)^\top$$

be the defender's and attacker's mixed strategies, respectively, satisfying

$$\mathbf{x}_i^* \geq 0, \quad \sum_{i=1}^{m} \mathbf{x}_i^* = 1, \qquad \mathbf{y}_j^* \geq 0, \quad \sum_{j=1}^{n} \mathbf{y}_j^* = 1.$$

Then the equilibrium v^* of the game is

$$v^* \;=\; \mathbf{x}^{*\top} P \, \mathbf{y}^* \;=\; \sum_{i=1}^{m} \sum_{j=1}^{n} \mathbf{x}_i^* \, P_{ij} \, \mathbf{y}_j^*.$$