A Game-Theoretic Framework for Detecting Advanced Persistent Threats

Dinuka Sahabandu

A dissertation
submitted in partial fulfillment of the
requirements for

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Radha Poovendran, Chair

Lillian Ratliff

James A. Ritcey

Program Authorized to Offer Degree:
Electrical and Computer Engineering

University of Washington

Abstract

A Game-Theoretic Framework for Detecting
Advanced Persistent Threats

Dinuka Sahabandu

Chair of the Supervisory Committee:
Radha Poovendran
Electrical and Computer Engineering

Advanced Persistent Threats (APTs) are stealthy and long-term attacks on cyber systems that threaten the security and privacy of sensitive information. The interactions of APTs with a victim system introduce information flows which are recorded in system logs. Dynamic Information Flow Tracking (DIFT) is a promising mechanism that examines the usage of information flows for detecting APTs. DIFT taints (tags) information flows originating at system entities that are susceptible to an attack, tracks the propagation of tainted flows, and authenticates the tainted flows at certain system components (traps) according to a pre-defined security policy. The deployment of DIFT to defend against APTs in cyber systems is limited by heavy resource and performance overhead. The effectiveness of detecting APTs depends on the False-Negatives (FNs) and False-Positives (FPs) associated with DIFT's security analysis. FNs and FPs of DIFT arise due to the inability of DIFT's pre-defined security policies to detect stealthy behavior of APTs.

In this dissertation, we use game theory to develop an interaction model between DIFT and APT. We use this model to study the trade-off between resource efficiency and the effectiveness of detection. Our game-theoretic framework incorporates several parameters that characterize the interaction, including costs of performing security analysis, false positives, and false negatives. We make use of the system log data information and postmortem/offline analysis of the data and construct an Information Flow Graph (IFG) to capture the victim system's events during the

execution time. We model and evaluate our game-theoretic frameworks on the IFGs extracted from a set of real-world attack datasets. We summarize the contributions of this dissertation below.

First, we consider simplified models for DIFT and APT in-order to draw insights on their interactions. Specifically, we assume DIFT does not incur any false-negatives and false-positives when performing a security analysis on a tagged information flow. The objective of DIFT is to identify the best set of system locations for tagging information flows that incur minimum resource overhead and enable high probability of APT detection via ensuring the tagged flows reach a set of predefined traps. We consider an APT whose goal is to evade detection and reach its target(s) in the victim system (single-stage attacks). We formulate a nonzero-sum, imperfect information game (DIFT-APT game) that models the interactions between the DIFT and APT. We characterize equilibrium strategies for both the defense and adversary, and design efficient algorithms for computing the strategies.

Then, we extend the DIFT-APT game model to incorporate multi-stage APT attacks where adversary sequentially passes through a set of intermediate targets in the system before reaching its final target. We model the best responses of the DIFT and APT using a shortest path problem and a submodular optimization problem, respectively. For a special case of the problem where the attack is a single-stage attack, we show a Nash equilibrium can be computed using a min-cut problem. We provide a polynomial-time algorithm to compute a correlated equilibrium for the multi-stage attack case. An equilibrium policy of DIFT identifies the best set of tag sources, trap locations, and pre-defined security rules that incur minimum resource cost while enabling high detection probability of an APT.

Next, we model the detection of multiple APTs using resource constrained DIFT. Given the attackers' strategies, we prove that finding an optimal defense strategy is equivalent to maximizing an increasing DR-submodular function. Given a defense strategy and strategies of other attackers, we show that finding an optimal attacker strategy is equivalent to solving a shortest path problem.

Then, we incorporate the false-negatives and false-positives associated with DIFT's security

analysis by modeling the strategic interactions between DIFT and APT as a stochastic game. We prove that the best response of the APT is a maximal reachability probability problem. We formulate the best response of the defense as a linear optimization problem. We present a nonlinear programming based polynomial-time algorithm to find an ϵ-Nash equilibrium (NE) of the discounted stochastic DIFT-APT game.

Next, we provide a model-free reinforcement learning algorithm to compute an NE of the discounted stochastic DIFT-APT game when the underlying false negatives and false positives of the DIFT are unknown. Specifically, we use an actor-critic algorithm that combines value-based and policy-based methods for faster convergence rates and smaller convergence errors.

Then, we formulate the interactions between DIFT and APT as an average reward stochastic game to capture the long-term behavior of the APTs. We show the existence of an Average Reward Nash Equilibrium (ARNE) in DIFT-APT game. We propose a reinforcement learning algorithm, RL-ARNE, to learn an ARNE of DIFT-APT game and prove the convergence of RL-ARNE algorithm to an ARNE of an average reward stochastic game.

Finally, we study the problem of Instruction Set Architecture (ISA) identification using partial binaries to facilitate DIFT in detecting known malicious patterns recorded in the program binaries. We use two different datasets with binaries from 12 ISAs and 23 ISAs to show that byte-level $(1, 2, 3)$-gram TF-IDF features yield high accuracy ($\sim 98\%$) compared to the existing byte-histogram and signature-based features ($\sim 91\%$). Additionally, we show that character-level $(1, 2, 3)$-gram TF-IDF features extracted from encoded binaries yield high accuracy with $16\times$ fewer features compared to the number of byte-level $(1, 2, 3)$-gram TF-IDF features.

For future research, we propose to investigate interpreting a Nash equilibrium of the average-reward stochastic game between DIFT-APT using graph-theory. Additionally, we provide insights on integrating signature and anomaly-based detection into our game framework. We believe this line of research is a promising endeavor to enable resource efficient DIFT for detecting APTs.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor Prof. Radha Poovendran and co-advisor Prof. Linda Bushnell for their invaluable guidance, support, and mentorship throughout the entire research process. Their expertise, encouragement, and constructive feedback have been instrumental in shaping this dissertation.

I am immensely grateful to the members of my dissertation committee, Prof. Lilian Ratliff, Prof. James Ritcey, and Prof. Arvind Krishnamurthy, for their valuable insights, thoughtful suggestions, and critical evaluation of my work. Their expertise and scholarly contributions have significantly enriched the quality of this research.

I would like to express my heartfelt gratitude to Prof. Wenke Lee, Prof. Sukarno Mertoguno, Prof. Andrew Clark, Prof. Shana Moothedath, Prof. Bhaskar Ramasubramanian, Dr. Luyao Niu, Dr. Baicen Xiao, and Dr. Joey Allen who generously shared their time and insights, making this research possible. Their contributions are greatly appreciated.

I would also like to thank Dr. Daniel Koller, program manager of Office of Naval Research Cyber Program for his valuable comments during the program review and acknowledge the support of ONR grants N00014-16-1-2710 P00002 and N00014-20-1-2636.

I am indebted to my family and friends for their unwavering support, understanding, and encouragement throughout this journey. Their love, belief in my abilities, and words of encouragement have been a constant source of motivation.

Last but not least, I would also like to extend my appreciation to the faculty and staff of University of Washington and the members in UW-NSL who provided an intellectually stimulating environment and resources that were essential for the completion of this dissertation.

DEDICATION

I dedicate this dissertation to my beloved family, whose unwavering love and support have been my guiding light throughout this academic journey.

To my wife, Nishaevithaa Somaskantha Iyer, I am forever grateful for your sacrifices, as you have selflessly stood by my side and provided the encouragement, comfort, and reassurance I needed during the most demanding times. Your unwavering love has been my anchor, grounding me and reminding me of what truly matters.

To my parents, Udaya Sahabandu and Lalani Liyanage, your endless sacrifices, encouragement, and belief in my abilities have been the driving force behind my accomplishments. Your unwavering support and unconditional love have given me the strength to pursue my dreams.

To my brother, Kalana Sahabandu, thank you for being my constant source of inspiration. Your unwavering faith in my abilities and your constant encouragement have motivated me to push beyond my limits.

To my advisors Prof. Radha Poovendran and Prof. Linda Bushnell, thank you for your guidance, wisdom, and expertise. Your guidance has shaped my academic growth and has opened my eyes to new perspectives and possibilities.

To the participants who generously gave their time and shared their experiences, thank you for your contribution to this research. Your insights have been invaluable in shaping the findings and conclusions of this study.

Finally, I dedicate this dissertation to all those who have paved the way in the fields of cyber security, game theory, and machine learning. Their groundbreaking research and contributions have inspired me to pursue knowledge and make my own contribution to the field.

Chapter 1

INTRODUCTION

Advanced Persistent Threats (APTs) are an emerging class of cyber threats that victimize governments and organizations around the world through cyber espionage and sensitive information hijacking [62, 137]. Unlike ordinary cyber threats (e.g., malware, Trojans) that execute quick damaging attacks, APTs employ sophisticated and stealthy attack strategies that enable unauthorized operation in the victim system over a prolonged period of time [131]. The ultimate objective of an APT typically aims to sabotage critical infrastructures (e.g., Stuxnet [49]) or exfiltrate sensitive information (e.g., Operation Aurora, Duqu, Flame, and Red October [24]). APTs follow a multi-stage approach to achieve the goal of the attack. Each stage of an APT is customized to exploit a set of vulnerabilities in the victim system to achieve a set of sub-goals (e.g., stealing user credentials, network reconnaissance) that will eventually lead to the end goal of the attack [140]. The stealthy, sophisticated and strategic nature of APTs makes detecting and mitigating their impact challenging using conventional security mechanisms such as firewalls, anti-virus software, and intrusion detection systems that rely heavily on the signatures of malware or anomalies observed in the benign behavior of the system.

Although APTs operate in a stealthy manner without inducing any suspicious abrupt changes, the interactions of APTs with the victim system introduce information flows. Information flows consist of data and control commands that dictate how data is propagated between different system entities (e.g., instances of a computer program, files, network sockets) [41,97]. Dynamic Information Flow Tracking (DIFT) is a mechanism developed to dynamically track the usage of information flows during program executions [97, 124]. Operation of DIFT is based on three steps. (i) Taint (tag) all the information flows that originate from the set of system entities susceptible to cyber threats [97], [124]. (ii) Propagate the tags into the output information flows based on a set of

predefined tag propagation rules which track the mixing of tagged flows with untagged flows at the different system entities. (iii) Verify the authenticity of the tagged flows by performing a security analysis at a subset of system entities using a set of pre-specified tag check rules. When a tagged (suspicious) information flow is verified as malicious through a security check, DIFT makes use of the tags of the malicious information flow to identify victimized system entities of the attack and reset or delete them to protect the system. Since information flows capture the footprint of APTs in the victim system and DIFT allows tracking and inspection of information flows, DIFT has been recently used as a defense mechanism against APTs [31], [47].

Tagging and tracking information flows in a system using DIFT adds additional resource costs to the underlying system in terms of memory and storage. In addition, inspecting information flows demands extra processing power from the system. Since APTs maintain the characteristics of their malicious information flows (e.g., data rate, spatio-temporal patterns of the control commands) close to the characteristics of benign information flows [135] to avoid detection, pre-defined security check rules of DIFT can miss the detection of APTs (false-negatives) or raise false alarms by identifying benign flows as malicious flows (false-positives). Typically, the number of benign information flows exceeds the number of malicious information flows in a system by a large factor. As a consequence, DIFT incurs a tremendous resource and performance overhead to the underlying system due to frequent security checks and false-positives. The high cost and performance degradation of DIFT can be worse in large scale systems such as servers used in data centers [47].

There have been approaches that perform system log data-based postmortem analysis to reduce the resource and performance cost of DIFT [31, 64]. However, widespread deployment of DIFT across various cyber systems and platforms is heavily constrained by the added resource and performance costs that are inherent to DIFT's implementation and due to false-positives and false-negatives generated by DIFT [63], [99]. An analytical model of DIFT needs to capture the system level interactions between DIFT and APTs, and cost of resources and performance overhead due to security checks. Additionally, false-positives and false-negatives generated by DIFT also need to be considered while deploying DIFT to detect APTs. At present, however, no such framework

exists.

A key property of APTs is that they operate in the victim system for long periods of time while achieving their malicious goals (e.g., data ex-filtration) [131], [36]. System logs record the behavior of the APTs in the system. *Record-replay* is a widely used offline mechanism used to detect APTs by postmortem analysis of system log data in batch-processing manner using DIFT [30, 65, 66, 74, 75, 78]. Our goal is to model the detection of APTs via postmortem analysis of system log data using DIFT. We provide game-theoretic models that enable the study of trade-off between resource efficiency and effectiveness of detection of DIFT. Strategic interactions of an APT to achieve the malicious objective while evading detection depends on the effectiveness of the DIFT's defense policy. On the other hand, determining a resource-efficient policy for DIFT that maximizes the detection probability depends on the nature of APT's interactions with the system. Non-cooperative game theory provides a rich set of rules that can model the strategic interactions between two competing agents (DIFT and APT). We capture the victim system's events during the execution time in a form of a graph called Information Flow Graph (IFG) constructed using the system log data information and postmortem/offline analysis of the data. We model our game-theoretic framework on the IFG of the system and propose data-driven algorithms to compute defense policies that maximize APT detection and minimize the resource cost of DIFT.

1.1 Summary of Research Contributions

A DIFT defense taints and tracks suspicious information flows across the network in order to identify possible attacks, at the cost of additional memory overhead for tracking non-adversarial information flows. Hence, in our initial modeling framework, we consider a DIFT defender whose goal is to identify the best set of system locations to tag information flows that incur minimum resource overhead and enable high probability of APT detection for a set of predefined system locations (traps) where tagged information flows are inspected for their authenticity. We assume any tagged adversarial information flow inspected at a trap location leads to perfect detection. We assume there are no false negatives and false positives associated with the DIFT's security analysis at the trap locations. We consider a single-stage APT whose goal is to evade detection

and reach its target(s) in the victim system. We formulate a nonzero-sum, imperfect information game (DIFT-APT game) that describes the interactions between the DIFT-based defense and APT's adversarial information flows. We develop efficient algorithms for computing the equilibrium strategies for DIFT and APT for the cases where APT has a single target and multiple targets in the victim system. We evaluate our approaches through a simulation study on a real-world dataset, ScreenGrab [65]. Chapter 3 presents the details of the game-theoretic model, the proposed algorithms and the simulation results, which appeared in our paper [116].

We extend the DIFT-APT game model to capture multi-stage APT attacks where adversary sequentially passes through a set of intermediate targets in the system before reaching its final target. Additionally, we consider a generalized DIFT-based defender model whose goal is to choose a set of best tag sources, trap locations, and pre-defined security rules that incur minimum resource and performance overhead while enabling high detection probability of an APT. We show the best response of an APT can be obtained as a solution to to a *shortest path problem* on a directed graph such that the shortest path corresponds to a path of maximum probability of reaching the final target. We exploit the *submodularity* property of the defender's payoff function to propose a polynomial time algorithm to find the best response of the defender with $1/2$-optimality guarantee. For the single-stage attacks, we characterize the set of *Nash equilibria* of the DIFT-APT game using a *min-cut problem*. We provide a polynomial-time iterative algorithm to compute a local *correlated equilibrium* of the game for the multi-stage attack. We simulate our game model and algorithm on real-world nation state attack dataset [65]. In Chapter 4 we detail the game framework and related results, which appeared in our paper [91].

We model simultaneous detection of K attackers in a system using DIFT by incorporating the different resource costs associated with detecting different types of attackers (e.g., APT vs. malware). We prove that finding an optimal defense strategy against given attackers' strategies is equivalent to maximizing an increasing *DR-submodular* function subject to a polytope constraint. Using the equivalence, we provide a polynomial-time algorithm to compute an $(1-1/e)$-approximate optimal strategy of the defender that provides the best known bound for submodular optimization. We show that finding an optimal attacker's strategy, for given strategies of other attackers and the

defender, is equivalent to solving a *shortest path problem*. We evaluate our approach on the day one attack data of the nation state attack [65]. Chapter 5 presents the details of the game theoretic model and the related results, which appeared in our paper [114].

We model the strategic interactions between DIFT and APT as a two-player nonzero-sum stochastic game to incorporate the false negatives and false positives associated with DIFT. We relate the best response of the APTs to a *maximum reachability probability problem* and formulate it as a linear program. We formulate the best response of the DIFT as a *linear program* and present a polynomial-time algorithm to calculate a deterministic optimal policy of the DIFT. We formulate discounted version of the game to perform approximate equilibrium analysis and present a value-iteration algorithm with guaranteed convergence and prove that the algorithm returns an ϵ-Nash equilibrium for the undiscounted game in polynomial-time. We evaluate our approach using NetRecon dataset [65]. In Chapter 6 we detail the game framework and the related results, which appeared in our paper [115].

We present a model-free, actor-critic, reinforcement learning algorithm to compute a Nash Equilibrium (NE) of the discounted stochastic DIFT-APT game formulated in Chapter 6. The proposed algorithm enables computing an NE of the game through simulated DIFT and APT on the system's IFG when the underlying false negatives and false positives of the DIFT are unknown. Actor-critic algorithms combine value-based and policy-based methods for faster convergence rates and smaller convergence errors [53, 73, 126]. We evaluate our approach using nation state attack dataset. Chapter 7 presents the details of the actor-critic algorithm and the related results, which appeared in our paper [113].

We model the interactions between DIFT and APT as an average reward stochastic game to capture the long-term behavior of APTs. The long-term behavior of APT provides DIFT with more opportunities to perform security analysis on adversarial information flows at multiple system locations. Therefore, long-term behavior of APTs can be exploited by the DIFT to increase the chances of successfully detecting APTs while minimizing the resource cost. We show that our model of the DIFT-APT game contains an an Average Reward Nash Equilibrium (ARNE). We take a data-driven approach to learn an ARNE of a nonzero-sum stochastic games called

RL-ARNE, an actor-critic-based reinforcement learning algorithm that learns an ARNE using Temporal Difference (TD) error minimization. We prove the convergence of RL-ARNE algorithm to an ARNE of the game using stochastic approximation. We evaluate the performance of our approach via an experimental analysis on real-world, large-scale attack data corresponding to ransomware and nation-state attacks implemented by US DARPA red-team during the evaluation of Refinable Attack INvestigation (RAIN) [65] system log recording system. Chapter 8 presents the details of the game model, RL-ARNE algorithm, and the related results.

DIFT's security rules often comprise of identifying control paths, data flows, and data types required for assessing content of program binary files (malicious/benign). This process requires knowledge of Instruction Set Architecture (ISA) of the binary file which is recorded in the file header. However, APTs have been observed to tamper with the meta-data fields in file headers to remain hidden during the attacks [44, 88]. Modern computer systems consist of large amount of multi-architecture binary files for initiating virtual machines and connecting to hardware peripherals such as GPUs, external storage devices, cameras, and printers. Hence, the absence of trustworthy ISA information in binary file headers thwart the DIFT's detection capabilities. Therefore, we study the problem of ISA identification using partial binaries to enhance DIFT's detection capabilities. We observe that successive bytes in binaries have co-occurring patterns specific to the ISA. We propose two N-gram TF-IDF-based binary code feature extraction methods to identify ISA from partial binaries. We evaluate the performance of our methods using two different datasets with binaries from 12 ISAs [9] and 23 ISAs [7]. Our experiments show that the proposed methods yield high accuracy ($\sim 98\%$) compared to the existing byte-histogram and signature-based features ($\sim 91\%$). Chapter 9 presents the details of the proposed binary code feature extraction methods and the related experimental results, which appeared in [112].

1.2 Related Work

1.2.1 Game Theory for Modeling Cyber Security Problems

Game theory has been widely used in the literature to analyze and design security in cyber systems against different types of adversaries [127], [13]. For instance, the FlipIt game in [132] captures all the interaction between APTs and the defender when both the players are trying to take control of a cyber system. In [132], APT and defender both take actions periodically and pay a cost for each of their action. Lee *et al.* in [76] introduced a control-theoretic approach to model competing malwares in the FlipIt game. Game models are available for APT attacks in cloud storage [90] and cyber systems [109]. The interaction between an APT and a defender that allocates Central Processing Units (CPUs) over multiple storage devices in a cloud storage system is formulated as a Colonel Blotto (zero-sum) game in [90]. Another zero-sum game model is given in [109] to model the competition between APT and defender in a cyber system.

Often in practice, the resource costs for the defender and the adversary are not the same, hence the game model is nonzero-sum. In this direction, a nonzero-sum game model is given in [61] to capture the interplay between the defender, the APT attacker, and the insiders for joint attacks. The approach in [61] models the incursion stage of the APT attack, while our model in this dissertation captures the lateral propagation stages of an APT attack. More precisely, we provide a multi-stage game model that detects APTs by implementing a data-flow-based DIFT detection mechanism while minimizing resource costs.

1.2.2 Stochastic Games

Stochastic games introduced by Shapley generalize Markov decision processes to model the strategic interactions between two or more players that occur in a sequence of stages [121]. Dynamic nature of stochastic games enables the modeling of competitive market scenarios in economics [14], competition within and between species for resources in evolutionary biology [52], resilience of cyber-physical systems in engineering [145], and secure networks under adversarial interventions in computer science [84].

Study of stochastic games is often focused on finding a set of *Nash Equilibrium* (NE) [94] policies for the players such that no player is able to increase their respective payoffs by unilaterally deviating from their NE policies. The payoffs of a stochastic game are usually evaluated under discounted or limiting average payoff criteria [50, 123]. Discounted payoff criteria, where future rewards of the players are scaled down by a factor between zero and one, is widely used in analyzing stochastic games as an NE is guaranteed to exist for any discounted stochastic game [89]. Limiting average payoff criteria considers the time-average of the rewards received by the players during the game [123]. The existence of an NE under limiting average payoff criteria for a general stochastic game is an open problem. When an NE exists, value iteration, policy iteration, and linear/nonlinear programming based approaches are proposed in the literature to find an NE [50, 106]. These approaches, however, require the knowledge of transition structure and the reward structure of the game. Also, these solution approaches are only guaranteed to find an exact NE only in special classes of stochastic games, such as zero-sum stochastic games, where rewards of the players sum up to zero in all the game states [50].

1.2.3 Multi-Agent Reinforcement Learning Algorithms

Multi-agent reinforcement learning (MARL) algorithms have been proposed in the literature to obtain NE policies of stochastic games when the transition probabilities of the game and reward structure of the players are unknown. MARL algorithms can be grouped into three categories based on the objectives of the players [144]: (i) Cooperative games where players coordinate to achieve a common goal. (ii) Competitive games where players compete against each other, and for any set of strategies the sum of the rewards to all players is zero (referred to as zero-sum stochastic games). (iii) Mixed games where each player tries to maximize its individual payoff function and the rewards of the players may not necessarily add up to zero (referred to as nonzero-sum stochastic games). In this dissertation we focus on mixed MARL algorithms since the interaction between DIFT and APT forms a nonzero-sum stochastic game (details in Section 7.2). A survey on MARL algorithms is presented in [144].

The authors of [79, 80] introduced a Q-learning algorithm (Nash-R) to learn an NE of average

reward stochastic games. Nash-R was *empirically* shown to find an NE of a nonzero-sum game by ensuring the players always use the same NE value for updating their Q-values. However, convergence guarantee of Nash-R requires an assumption that the game has a unique NE value. The DIFT-APT game that we study in this dissertation has nonzero-sum payoff structure due to the resource costs incurred by DIFT in performing security analysis. In general, nonzero-sum games have been observed to have multiple NE values [50]. Nash-R also requires solving a matrix game corresponding to a state of the game at each iteration of the algorithm which is PPAD-complete [38, 45] and incurs a memory complexity that is exponential in number of players for storing the Q-tables.

Q-learning algorithms proposed in [60, 82] for discounted stochastic games require solving matrix games and similar conditions as in Nash-R for the convergence. To enhance the scalability of MARL algorithms for nonzero-sum *discounted* stochastic games, recent works in [16, 103, 104] developed actor-critic algorithms. The convergence of the algorithm in [16] was guaranteed for weakly acyclic games. The algorithms in [103, 104] used minimization of temporal difference (TD) error i.e., Bellman residual error, and provided guarantees on the convergence to an NE using stochastic approximation. While our work also uses TD error minimization and stochastic approximation, our focus is on designing a scalable MARL algorithm for nonzero-sum *average reward* stochastic games. Recent work in [86, 101, 118] present efficient MARL algorithms for learning NE in *zero-sum* stochastic games. However, these algorithms are not applicable to *nonzero-sum* DIFT-APT game that we study in this dissertation.

1.3 Dissertation Outline

The remainder of this dissertation is organized as follows: Chapter 2 presents the preliminaries about DIFT, IFG, and APT models considered in this dissertation. Chapter 3 presents DIFT-APT game model for detecting single-stage cyber attacks. Chapter 4 presents DIFT-APT game model for detecting multi-stage cyber attacks. Chapter 5 presents DIFT-APT game model for simultaneously detecting multiple single-stage cyber attacks. Chapter 6 presents stochastic DIFT-APT game model. Chapter 7 presents a reinforcement learning algorithm for computing a Nash Equilibrium of discounted stochastic DIFT-APT game. Chapter 8 presents average reward stochastic DIFT-APT game model and RL-ARNE algorithm for computing average reward Nash equilibrium policies of stochastic games. Chapter 9 presents natural language processing-based feature extraction methods for instruction set architecture identification using partial binaries. We conclude this dissertation and present promising future directions in Chapter 10.

Chapter 2

PRELIMINARIES

2.1 Dynamic Information Flow Tracking (DIFT)

DIFT is a taint analysis system that dynamically monitor the operation of a system. It consists of three components: (i) taint sources, (ii) taint propagation rules, and (iii) taint sinks. Taint (tag) sources are processes and objects in the system that are considered as *untrusted* sources of information, i.e., λ. All the information flows originating from a taint source are labeled or tagged and then its use is dynamically tracked using the taint propagation rules [134]. Taint propagation rules define how to propagate tags into the output information flows when tagged flows are used with untagged flows at the system processes and objects. Finally the tagged flows undergo security analysis at dynamically generated security check points called as *taint sinks (traps)* when an unauthorized use is observed. Tag propagation rules and tag check rules at the traps are defined by systems security experts and often called as the security policy of the DIFT. When a tagged (suspicious) information flow is verified for its unauthorized use at a trap process, DIFT marks it as a malicious flow and trace back to its entry point in the system to terminate the victimized process.

2.2 Information Flow Graph

An *information flow graph* (IFG) is a directed multigraph that represents the history of a system's execution in terms of the spatio-temporal relationships between processes and objects (files and network endpoints) [64]. Processes and objects are nodes in the graph and the directed edges describe interactions and information flows between the nodes. Using provenance-enhanced auditing is heavily desired by large enterprises and government agencies due to its ability to answer two key questions, how an attack infiltrated their systems and what are the ramifications of the attack.

Unfortunately, classical auditing systems cannot efficiently answer these questions; this is because they cannot effectively embed the causal relationships into uniform records, like provenance-enhanced systems. When causal relationships are embedded into audit logs, security-experts can run provenance-dependent queries to derive the origin of an attack and the ramifications of an attack. Identifying the origins of an attack is completed by doing a backward traversal, which analyzes the ancestral dependencies of the attack. Additionally, forward analysis techniques traverse through the graph in the forward direction, which effectively determines the ramifications of the attack [64]. In order to detect APTs we use the IFG obtained from the system log.

Let $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}})$ represents the IFG of the system. $V_{\mathcal{G}} = \{s_1, \ldots, s_N\}$ consists of the processes (e.g., an instance of a computer program) and objects in the system and $E_{\mathcal{G}} \subseteq V_{\mathcal{G}} \times V_{\mathcal{G}}$ represents the information flows (directed) in the system from one node to the other. We model our game-theoretic framework on the IFG of the system to capture the strategic interactions between DIFT and APT.

2.3 Advanced Persistent Threats (APTs)

Advanced persistent threats (APTs) are sophisticated attackers, such as groups of experienced cybercriminals, that establish an illicit, long-term presence in a system in order to mine valuable information/intelligence. The targets of APTS, which are very specifically chosen, typically include large enterprises or governmental networks. The attacker spends time and resources to identify the vulnerabilities that it can exploit to gain access into the system, and to design an attack that will likely remain undetected for a long period of time. These attacks are stealthy and differ from the conventional cyber attacks in complexity and their ability to evade the intrusion detection systems by adopting a nominal system behavior. We can break down a successful APT campaign into the following key stages:

1. Initial Compromise: During the initial compromise stage, the attacker's goal is to gain access to an enterprise's network. In most cases, the attacker achieves this by exploiting a vulnerability or a social engineering trick, such as a phishing email.

2. Foothold Establishment: Once the attacker has completed the initial compromise, it will establish a persistent presence by opening up a communication channel with their Command & Control (C&C) server.

3. Privilege Escalation: Next, the attacker will try and escalate its privileges which may be necessary in order to access sensitive information, such as proprietary source code or customer information.

4. Internal Reconnaissance: During the reconnaissance phase, the attacker will try to gain information about the system, such as what nodes are accessible on the system and the security defenses, such as an IDS, that are being used.

5. Lateral Movement: The attacker will increase its control of the system by moving laterally to new nodes in the system.

6. Attack Completion: The final goal of the attacker is to deconstruct the attack, hopefully in a way to minimize his footprint in order to evade detection. For example, attackers may rely on removing the system's log.

Let $\lambda \subset V_{\mathcal{G}}$ be the possible *entry points* of the adversary and $\mathcal{D}_j \subset V_{\mathcal{G}}$ be the set of targets, refereed to ad *destinations*, of stage j of the attack.

Chapter 3

DIFT-APT GAMES FOR DETECTING SINGLE-STAGE CYBER THREATS

3.1 Motivation

The idea behind DIFT is to tag (taint) untrusted information flows and track their propagation at byte or word-granularity through the system. Use of DIFT across a large-scale system, however, also involves tainting and tracking a large number of valid information flows, imposing a significant overhead in memory consumption as additional bits must be allocated to each memory address for tainting. In [48] it has been shown that tainting all information flows can incur 2-20 times slowdown for the system. In order to reduce this performance overhead and enable widespread deployment of DIFT, one possible approach is to only taint a subset of information flows that pass through specific system processes. The choice of which processes to taint will depend on which processes are likely to be traversed by adversarial information flows, which will in turn depend on the interaction between the adversary and the targeted system.

An analytical model of DIFT and its interaction with adversarial information flows would enable evaluation of the performance cost and effectiveness of flow-tainting strategies, as well as design of optimal tainting policies to enable resource-efficient DIFT. At present, however, no such framework exists.

In this chapter we present such an analytical modeling framework for DIFT. Our framework is based on the following insights. First, the effectiveness of the defense depends on the adversary's strategy, while the adversary's probability of avoiding detection will be determined by the defense policy. This strategic interaction motivates a game-theoretic approach. Second, the game unfolds at multiple stages as the attack transitions between different system processes between the source and destination of the information flow. We therefore formulate a game model that is played

on a system call graph that describes the feasible transitions between processes. At each stage, the adversary decides which process to transition to at the next stage, while the defense decides whether to taint the information flow or not, at the cost of performance overhead due to tainting of valid flows that pass through the same process. We make the following specific contributions:

- We formulate a game describing the interaction between the DIFT-based defense and an adversarial information flow. The game describes the goals of both adversary and defense, as well as the information asymmetry between the players, and is valid for any APT that can be detected via DIFT.

- In the case where there is a single destination for the attack that is known to both players, we characterize the equilibrium stationary strategies of both players.

- In the case where there are multiple destinations with different values for the attacker, we derive an efficient algorithm for computing the equilibrium stationary strategies for both players.

- We evaluate our approach through simulation study on a real-world dataset describing the ScreenGrab attack, which was gathered using the Refinable Attack Investigation (RAIN) system [65]. Our simulation study provides insight into the predicted behaviors of both adversary and defense under our model, as well as where traps should be located.

We observe that, while we focus on equilibrium strategies in this chapter, sudden changes in the cyber environment or adversary capabilities may prevent the interaction from reaching an equilibrium. The information flows, however, will persist over a longer period of time even after such changes occur, making equilibrium a meaningful concept for this class of games.

3.2 DIFT-APT Game Formulation

The game unfolds in a series of stages $t = 1, 2, \ldots$. The set of game states is defined as follows. Let $\mathcal{S}$ denote the set of nodes in the provenance graph [65]. The state space is given by $\overline{\mathcal{S}} = \mathcal{S} \times \{0, 1\}$.

The 0 represents the case where the flow has not been tainted, while 1 represents the case where the flow has been tainted. We write $\overline{s}_t = (s_t, b_t)$ where $s_t \in \mathcal{S}$ and $b_t \in \{0, 1\}$. We let $\mathcal{T}$ denote a set of *trap locations*.

The set of actions are defined as follows. For the DIFT system (i.e. the defender), the set of actions at a state $\overline{s} = (s, 0)$ is given by $A_1(\overline{s}) = \{0, 1\}$, i.e., the defender can choose to tag or not tag the flow at that point. At a state $\overline{s} = (s, 1)$, the set of actions is $A_1(\overline{s}) = \emptyset$ and the state $b_t = 1$ for all subsequent steps. Once the flow is tagged as a spurious flow, DIFT keeps track of the tagged flow and this will incur memory and performance overhead for the defender [65]. For the adversary, the set of actions $A_2(\overline{s})$ is a subset of S, and represents the next state that is reached by the flow. For a state $s \in \mathcal{S}$, the set of feasible next states is denoted $N(s)$. The adversary can also end the game by choosing to transition to a null state $\emptyset$, corresponding to dropping the information flow.

The state transitions are defined as follows. Let $\overline{s}_t$ denote the state at stage t, and let $a_1(t), a_2(t)$ denote the actions of the defender and adversary at the stage t, respectively. Then the state at the stage $(t + 1)$ will be given by

$$\overline{s}_{t+1} = \begin{cases} (a_2(t+1), a_1(t+1)), & \overline{s}_t = (s, 0) \text{ for some } s \\ (a_2(t+1), 1), & \overline{s}_t = (s, 1) \text{ for some } s \end{cases}$$

There are N destinations $\mathcal{D} = \{d_1, \ldots, d_N\}$ for the adversary's information flow. The game terminates when one of the following conditions hold: (i) the flow reaches a state $(s, 1) \in \mathcal{T} \times \{1\}$; (ii) the flow reaches a state $\overline{s} \in \mathcal{D} \times \{0, 1\}$ (if $d_i \notin \mathcal{T}$) or $\overline{s} = (d_i, 0)$ (if $d_i \in \mathcal{T}$); or (iii) the adversary chooses to transition to the null state. Condition (i) represents the case where the adversary reaches a trap and is caught, condition (ii) represents the case where the adversary reaches its destination, and condition (iii) represents the case where the adversary drops out of the game. We let T denote a random variable equal to the time when the game terminates.

The player utilities are defined as follows. The defender utility has three components, namely, a memory cost associated with tagging flows, a cost $\beta_D^i < 0$ incurred if the adversary successfully reaches the destination d_i, and a benefit $\alpha_D > 0$ for catching the adversary. We assume that the

cost of tagging flows is independent of the adversary's strategy, since it is assumed that most flows are benign, and hence the memory cost is dominated by the cost of tagging valid flows. This cost is therefore only incurred at the first state when tagging occurs at a time denoted t_{tag}, and is denoted $C_{\mathcal{D}}(s_{t_{tag}})$; note that the cost can depend on the state when tagging begins. The defender utility is therefore given by

$$U_{\mathcal{D}} = C_{\mathcal{D}}(s_{t_{tag}}) + V_{\mathcal{D}}(\overline{s}_T), \tag{3.1}$$

where

$$V_{\mathcal{D}(\overline{s}_T)} = \begin{cases} \beta_D^i, & \overline{s}_T = d_i, d_i \notin \mathcal{T} \\ & \text{or } \overline{s}_T = (d_i, 0), d_i \in \mathcal{T} \\ \alpha_D, & \overline{s}_T \in \mathcal{T} \times \{1\} \\ 0, & \text{else} \end{cases}$$

The adversary utility consists of a benefit $\beta_A^i > 0$ for successfully reaching the destination d_i and a cost $\alpha_A < 0$ if the adversary is tagged and trapped. We assume without loss of generality that $\beta_A^1 > \cdots > \beta_A^N$. The adversary's utility function is given by $U_{\mathcal{A}} = V_{\mathcal{A}}(\overline{s}_T)$, where $V_{\mathcal{A}}$ is defined analogously to $V_{\mathcal{D}}$ with parameters β_A^i and α_A.

The players also have different information sets. The defender knows the full state $\overline{s}_t$, while the adversary does not know b_t, i.e., the adversary does not know whether the flow has been tainted or not.

We observe that the set of possible strategies for the defender and adversary can be very large, due to the fact that both players can use information obtained from previous game stages (i.e., the previous state transitions) to determine their action at each stage. To reduce complexity and model the fact that both tagging and adversarial information flow are lower-level processes with limited computation capability, we restrict the set of strategies to stationary strategies, defined as follows.

Definition 1. *A player strategy is* stationary *if it depends only on the current state.*

A stationary strategy by the defender can be represented by a set of probabilities $\{p(s) : s \in \mathcal{S}\}$,

representing the probability that the defender will tag a particular flow at node s if it has not been tagged already. A stationary strategy by the adversary can be represented by a set of probabilities $\{p_A(s,s') : s \in \mathcal{S}, s' \in N(s) \cup \{\emptyset\}\}$, representing the probability of transitioning from node s to node s'. We define $\mathbf{p} = \{p(s) : s \in \mathcal{S}\}$, $\mathbf{p}_A = \{p_A(s,s') : s \in \mathcal{S}, s' \in N(s) \cup \{\emptyset\}\}$, and let $U_D(\mathbf{p}, \mathbf{p}_A)$ (resp. $(U_A(\mathbf{p}, \mathbf{p}_A)$) denote the utility to the defender (resp. attacker) arising from the strategies $(\mathbf{p}, \mathbf{p}_A)$. The solution concept of the game is defined as follows.

Definition 2. *Let the best response sets $BR(\mathbf{p})$ and $BR(\mathbf{p}_A)$ be defined by*

$$\begin{aligned} BR(\mathbf{p}) &= \arg\max_{\mathbf{p}_A} \{U_A(\mathbf{p}, \mathbf{p}_A)\} \\ BR(\mathbf{p}_A) &= \arg\max_{\mathbf{p}} \{U_D(\mathbf{p}, \mathbf{p}_A)\} \end{aligned}$$

A solution (equilibrium) to the game is a pair of stationary strategies $\mathbf{p}^*$, $\mathbf{p}_A^*$ such that

$$\mathbf{p}_A^* \in BR(\mathbf{p}^*), \quad \mathbf{p}^* \in BR(\mathbf{p}_A^*).$$

For nonzero-sum, imperfect information games, solving for optimal strategies as in Definition 2 is computationally difficult. In what follows, we focus on solving for *locally optimal* solutions, in which neither the defender nor the adversary can improve its utility by changing its strategy at any single state.

3.3 Solution to DIFT-APT Game

In this section, we provide a solution framework for the flow tracking game. We first focus on the special case where there is only a single destination, and then propose an algorithm for computing the equilibrium when there are multiple destinations.

3.3.1 Solution to Single-Destination Case

We first consider the case where there is only a single destination, and let d denote the destination throughout. While we assume that there is a single destination for ease of exposition, this approach is also valid if there are multiple destinations and all have the same value of $\beta_A^i \equiv \beta_A$. We define

the additional notation $U_{\mathcal{D}}(s)$ (resp. $U_{\mathcal{A}}(s)$) to denote the utility of the defender (resp. adversary) when the game starts in state $(s, 0)$. By definition, for stationary policies p and p_A, the utilities satisfy

$$\begin{aligned} U_{\mathcal{D}}(s) &= p(s)C_{\mathcal{D}}(s) + p(s)r(s)\alpha_D + p(s)w(s)\beta_D \\ &\quad +(1-p(s))\sum_{s'\in N(s)} p_A(s,s')U_{\mathcal{D}}(s') \\ U_{\mathcal{A}}(s) &= p(s)r(s)\alpha_A + p(s)w(s)\beta_A \\ &\quad +(1-p(s))\sum_{s'\in N(s)} p_{\mathcal{A}}(s,s')U_{\mathcal{A}}(s') \end{aligned}$$

where $r(s)$ denotes the probability that a flow originating at the source s will reach a trap before reaching its destination, and $w(s)$ is the probability that a flow originating at s will reach its destination before reaching a trap. We have that $r(s) = 1$ if s is a trap and 0 if s is a destination. Otherwise

$$r(s) = \sum_{s'} p_A(s,s')r(s').$$

Similarly, $w(s) = 1$ if s is the destination and is 0 if s is a trap. Otherwise

$$w(s) = \sum_{s'} p_A(s,s')w(s).$$

The functions $U_{\mathcal{D}}$ and $U_{\mathcal{A}}$ have boundary conditions given by $U_{\mathcal{D}}(s) = V_{\mathcal{D}}(s)$ and $U_{\mathcal{A}}(s) = V_{\mathcal{A}}(s)$ at the terminating states of the game (destination and traps).

The following lemmas allow us to reduce the space of possible strategies further.

Lemma 1. *Suppose that there is a path from s to d that does not pass through a trap. Let $(s_0, s_1), (s_1, s_2), \ldots, (s_{m-1}, s_m)$, with $s_0 = s$ and $s_m = d$, denote the path. Then the optimal strategy for the adversary at a state s is to choose a policy with $p_A(s_i, s_{i+1}) = 1$. The resulting utility is $U_{\mathcal{A}}(s) = \beta_A$.*

Lemma 1 states that if the adversary can avoid traps altogether, then the adversary will choose a path that avoids traps and win the game with probability 1. The proof is straightforward.

Lemma 2. *For any equilibrium strategies p^* and p_A^*, $p^*(s) < 1$ for all s.*

Proof. If $p(s) = 1$, then the adversary's utility is given by $r(s)\alpha_A + w(s)\beta_A$. If there is a path from s to the destination such that the adversary avoids all traps, then $r(s) = 0$, $w(s) = 1$, and the adversary will collect utility β_A, while the defender will have utility $C_{\mathcal{D}}(s) + \beta_{\mathcal{D}}$. The defender can then increase its utility by decreasing p, contradicting the assumption that p^* is optimal.

On the other hand, suppose that there is no path such that the adversary avoids all traps. Then the adversary's utility is $\beta_{\mathcal{A}} < 0$, implying that the adversary can improve its utility by choosing the null state and dropping out of the game. Hence all equilibrium strategies will have $p_A(s, \emptyset) = 1$. If $p_A(s, \emptyset) = 1$, however, the defender can improve its utility by reducing $p(s)$, again contradicting the assumption that $p^*(s) = 1$ is optimal. □

Motivated by Lemma 1, we define $\hat{\mathcal{S}} \subseteq \mathcal{S}$ to denote the set of states such that the adversary will have utility β_A, i.e., the adversary is guaranteed to reach the destination undetected with probability 1, under all equilibrium policies. Equivalently $w(s) = 1$ (and hence $r(s) = 0$) for all $s \in \hat{\mathcal{S}}$. By definition, $\hat{\mathcal{S}}$ contains d, provided $d \notin \mathcal{T}$.

By Lemma 1, the set $\hat{\mathcal{S}}$ contains the set of states that are reachable to d in the residual graph obtained by removing all nodes from $\mathcal{T}$. We let $\delta\hat{\mathcal{S}} = \{s : N(s) \cap \hat{\mathcal{S}} \neq \emptyset\}$. The following proposition characterizes the Nash equilibrium stationary strategies at a state $s \in \delta\hat{\mathcal{S}}$. As a preliminary, let $q(s) = 1 - p_A(s, \emptyset)$.

proposition 1. *Suppose that s is a state such that $U_A(s'') = \beta_A$ for some $s'' \in N(s)$. Then there is a unique equilibrium strategy for the defender and adversary given by*

$$(p^*(s), q^*(s)) = \begin{cases} \left(\frac{\beta_A}{\beta_A - \alpha_A}, \frac{C_{\mathcal{D}}(s)}{\beta_D - \alpha_D}\right), & \frac{C_{\mathcal{D}}(s)}{\beta_D - \alpha_D} \in (0,1) \\ (0,1), & \textit{else} \end{cases} \tag{3.2}$$

and $p_A^*(s, s'') = q^*(s)$.

Proof. The utility of the adversary is given by

$$\begin{aligned} U_A(s) &= p(s)r(s)\alpha_A + p(s)w(s)\beta_A \\ &\quad +(1-p(s))\sum_{s'} p_A(s,s')U_A(s') \\ &= p(s)q(s)\alpha_A + (1-p(s))\sum_{s'} p_A(s,s')U_A(s') \end{aligned}$$

We have that $U_A(s') \leq \beta_A = U_A(s'')$, and hence the adversary's optimal strategy is to move to state s' with probability $q(s)$ and to exit the game with probability $(1-q(s))$. Furthermore, since $s \notin \hat{\mathcal{S}}$, each path from s to d contains at least one trap, and hence the flow will be detected if the adversary transitions to state s'' and will be undetected if the flow is dropped at s. The adversary's utility can then be written as

$$\begin{aligned} U_A(s) &= p(s)q(s)\alpha_A + (1-p(s))q(s)\beta_A & (3.3) \\ &= q(s)(p(s)(\alpha_A - \beta_A) + \beta_A) & (3.4) \end{aligned}$$

Similarly, the defender's utility can be written as

$$\begin{aligned} U_D(s) &= p(s)C_{\mathcal{D}}(s) + p(s)q(s)\alpha_D + (1-p(s))q(s)\beta_D \\ &= p(s)(C_D(s) + q(s)(\alpha_D - \beta_D)) + q(s)\beta_D. \end{aligned}$$

By Lemma 2, $p^*(s) < 1$ for any equilibrium policy. In order to have $p^*(s) \in (0,1)$, we must have $C_{\mathcal{D}}(s) + q(s)(\alpha_D - \beta_D) = 0$, and hence $q^*(s) = \frac{C_{\mathcal{D}}(s)}{\beta_D - \alpha_D}$. Since $C_D(s) < 0$, $\beta_D < 0$ and $\alpha_D > 0$, this value is positive, although it may exceed 1.

Now, considering the adversary's utility, we have that $q^*(s) \in (0,1)$ only if $p^*(s) = \frac{\beta_A}{\beta_A - \alpha_A}$. Since $\beta_A > 0$ and $\alpha_A < 0$, this value is in the interval $(0,1)$.

Combining these, we have that there is an equilibrium $\left(\frac{\beta_A}{\beta_A - \alpha_A}, \frac{C_{\mathcal{D}}(s)}{\beta_D - \alpha_D}\right)$ if $\frac{C_D(s)}{\beta_D - \alpha_D} < 1$. If this result does not hold, then the optimal strategy of the defender is to set $p^*(s) = 0$, and hence the equilibrium is $(p^*(s), q^*(s)) = (0,1)$. □

Proposition 1 suggests the following procedure for computing the set $\hat{\mathcal{S}}$. For each $s \in \delta\hat{\mathcal{S}}$, compute the Nash equilibrium $(p^*(s), q^*(s))$. If $p^*(s) = 0$, then $s \in \hat{\mathcal{S}}$. The procedure terminates when no further states can be added to $\hat{\mathcal{S}}$. A pseudocode description is given as Algorithm 7.

Theorem 1. *The set of equilibria over stationary strategies is equal to the set of possible outputs of Algorithm 7.*

Proof. By Lemma 1, all nodes in $\hat{\mathcal{S}}_1$ at Line 3 have adversary utility β_A and $p^*(s) = 0$. Hence, by induction and Proposition 1, the Nash equilibrium strategy for each $s \in (\delta\hat{\mathcal{S}}_1 \setminus \hat{\mathcal{S}}_0)$ is given by Eq. (3.2). In particular, each state in $\hat{\mathcal{S}}_1$ has adversary utility β_A and $p^*(s) = 0$. All nodes in $\hat{\mathcal{S}}_0$ have adversary utility 0.

Due to the termination condition of the while loop in Lines 6-16, the set $\hat{\mathcal{S}}_0$ forms a cutset between $\mathcal{S} \setminus (\hat{\mathcal{S}}_1 \cup \hat{\mathcal{S}}_0)$ and $\hat{\mathcal{S}}_1$. Hence $U_A(s) = 0$ for all $s \in \mathcal{S} \setminus (\hat{\mathcal{S}}_1 \cup \hat{\mathcal{S}}_0)$. At each such state s, for the adversary, $p_A(s, s') = 0$ for all s' if $p^*(s) > 0$, and $p_A(s, s')$ is arbitrary when $p^*(s) = 0$. Thus $p^*(s) = p^*_A(s, s') = 0$ for all s' is the unique Nash equilibrium at such states. □

3.3.2 Solution with Multiple Destinations

We now consider a generalized game when there are multiple destinations $d_1, \ldots, d_M$, each with an associated benefit to the adversary β^i_A corresponding to d_i. The generalized game can be solved by the procedure shown as Algorithm 2. The approach of the algorithm is to maintain a set $\overline{\mathcal{S}}$ of states that have their strategies chosen. At each iteration, a state is added to $\overline{\mathcal{S}}$ by finding the state that provides the maximum utility $U_A(s)$ for the local game with utility functions

$$\begin{aligned} U_A(s) &= \alpha_A r(s)p(s) + \sum_i w_i(s)\beta^i_A p(s) \\ &\quad +(1-p(s))\sum_{s'\in\overline{\mathcal{S}}} p_A(s,s')U^*_A(s') \\ U_D(s) &= p(s)C_D(s) + p(s)\alpha_D r(s) + \sum_i w_i(s)\beta^i_D p(s) \\ &\quad +(1-p(s))\sum_{s'\in\overline{\mathcal{S}}} p_A(s,s')U^*_D(s') \end{aligned}$$

where $w_i(s)$ is equal to the probability that d_i is reached before any trap, and the probabilities $r(s)$ and $w_i(s)$ are computed by assuming that all states in $\overline{\mathcal{S}}$ follow their equilibrium policies and all states $s' \in (\mathcal{S} \setminus (\overline{\mathcal{S}} \cup \{s\})$ have $p_A(s', \emptyset) = 1$.

Algorithm 1 Algorithm for computing equilibrium in single-destination case.

procedure EQUILIBRIUM_COMPUTATION($G = (\mathcal{S}, E)$, α_A, β_A, α_D, β_D, $\mathcal{T}$, d)
  Input: Provenance graph G of states $\mathcal{S}$ and edges E, parameters α_A, α_D, β_A, β_D, set of traps $\mathcal{T}$, destination d
  Output: Sets $\hat{\mathcal{S}}_0$, $\hat{\mathcal{S}}_1$, optimal strategies $p^*(s) : s \in \mathcal{S}$, $p_A^*(s, \cdot) : s \in \mathcal{S}$
  $\hat{\mathcal{S}}_1 \leftarrow \mathcal{D} \setminus \mathcal{T}$
  $\hat{\mathcal{S}}_1 \leftarrow$ set of nodes connected to $\hat{\mathcal{S}}_1$ in graph with traps removed
  $\hat{\mathcal{S}}_0 \leftarrow \emptyset$
  $found \leftarrow 1$
  while $found == 1$ and $\delta\hat{\mathcal{S}}_1 \setminus \hat{\mathcal{S}}_0 \neq \emptyset$ **do**
    $found \leftarrow 0$
    for $s \in (\delta\hat{\mathcal{S}}_1 \setminus \hat{\mathcal{S}}_0)$ **do**
      Solve for equilibrium at s using Proposition 1 to obtain $p^*(s)$, $p_A^*(s, \cdot)$
      if $p^*(s) == 0$ **then**
        $\hat{\mathcal{S}}_1 \leftarrow \hat{\mathcal{S}}_1 \cup \{s\}$, $found \leftarrow 1$
      else
        $\hat{\mathcal{S}}_0 \leftarrow \hat{\mathcal{S}}_0 \cup \{s\}$
      end if
    end for
  end while
  for $s \in \mathcal{S} \setminus (\hat{\mathcal{S}}_1 \cup \hat{\mathcal{S}}_0)$ **do**
    $p^*(s) \leftarrow 0$
    $p^*(s, s') \leftarrow 0 \; \forall s' \in \mathcal{S}$
  end for
end procedure

Algorithm 2 Algorithm for computing equilibrium with multiple destinations.

procedure GENERALIZED_COMPUTATION
  $\overline{\mathcal{S}} \leftarrow \{d_1\}$
  while $\overline{\mathcal{S}} \neq \mathcal{S}$ **do**
    for $s \in \mathcal{S} \setminus \overline{\mathcal{S}}$ **do**
      Solve local game at s treating $\overline{S}$ strategies as given, obtain utility $\hat{U}_A(s)$
    end for
    $s^* \leftarrow \arg\max \{\hat{U}_A(s) : s \in \mathcal{S} \setminus \overline{\mathcal{S}}\}$
    $(p^*(s^*), p_A^*(s^*, \cdot)) \leftarrow$ equilibrium strategies of local game
    $\overline{\mathcal{S}} \leftarrow \overline{\mathcal{S}} \cup \{s^*\}$
  end while
end procedure

Before proving optimality of Algorithm 2, we let $(\mathbf{p}^*, \mathbf{p}_A^*)$ denote an equilibrium of the multi-destination game, and define a graph $\mathcal{G}^*$ with node set $\mathcal{S}$, where there is an edge $(s, s') \in \mathcal{G}^*$ if $p_A^*(s, s') > 0$. We have the following preliminary results on this graph.

Lemma 3. *For any stationary equilibrium policy, there exists an equilibrium* $(\mathbf{p}^*, \mathbf{p}_A^*)$ *such that the graph* $\mathcal{G}^*$ *is directed and acyclic, and for any* s *and* s', $(s, s') \in \mathcal{G}^*$ *implies that* $U_A(s) \leq U_A(s')$.

Proof. First, we have that

$$U_A(s) \quad = \quad \sum_{s'} p_A(s, s') \left[\alpha_A r(s') p(s) + \sum_i w_i(s') \beta_A^i p(s) + (1 - p(s)) U_A(s') \right].$$

At equilibrium, $p_A(s, s') > 0$ if and only if s' is a maximizer of $\alpha_A r(s') p(s) + \sum_i w_i(s') \beta_A^i p(s) + (1 - p(s)) U_A(s^*)$. Hence we have

$$U_A(s) \leq \alpha_A r(s') p(s) + \sum_i w_i(s') \beta_A^i p(s) + (1 - p(s)) U_A(s') \leq U_A(s')$$

Now, to see that the graph is directed and acyclic, the previous derivation implies that all states in the cycle must have the same value of U_A at equilibrium.

From the above equations, we have that

$$U_A(s) = p(s)(\alpha_A r(s^*) + \sum_i w_i(s^*)\beta_A^i - U_A(s^*)) + U_A(s^*).$$

We have that $U_A(s) = U_A(s')$ if either $p(s) = 0$ or $\alpha_A r(s^*) + \sum_i w_i(s^*)\beta_A^i = U_A(s^*)$. The left-hand side is equal to the adversary's utility at s^* if the flow has been tagged, while the right-hand side is equal to the adversary's utility at s^* if the flow has not been tagged. These quantities can only be equal if the flow avoids the set of traps with probability 1. If this is the case, then a new equilibrium policy with the same U_A can be constructed by either following a shortest path to a destination with probability 1 (removing the cycle), or dropping the flows with probability 1 for each state in the cycle. In both cases the cycle is removed and the utility of the adversary is maintained. □

proposition 2. *Algorithm 2 returns a stationary equilibrium of the game if such an equilibrium exists.*

Proof. The proof is by induction. We show that, at each iteration, the strategies at the states in $\overline{\mathcal{S}}$ are the strategies at an equilibrium. Since the maximum achievable utility is β_A^1, the result holds at the first iteration. Let s^k denote the state added to $\overline{\mathcal{S}}$ at the k-th iteration. Consider the graph $\mathcal{G}^*$ defined above, and suppose that $\mathcal{G}^*$ is directed and acyclic by Lemma 3.

We observe that there must be a path from s^k to a node $\hat{s} \in \mathcal{S} \setminus \overline{\mathcal{S}}$ such that all neighbors of $\hat{s}$ are in $\overline{\mathcal{S}}$. To see this, note that we can pick a sequence of states with $s_0 = s^k$, $(s_i, s_{i+1}) \in \mathcal{G}^*$, and $s_i \notin \overline{\mathcal{S}}$. Since $\mathcal{G}^*$ is directed and acyclic, each state can appear at most once in the path, and hence eventually we reach a state with no neighbors outside $\overline{\mathcal{S}}$. Since s^k is connected to $\hat{s}$, we must have $U_A(s^k) \leq U_A(\hat{s})$. On the other hand, s^k is the state where $U_A(s^k)$ is maximized over all strategies that assign positive probability only to states in $\overline{\mathcal{S}}$, and hence $U_A(s^k) \geq U_A(\hat{s})$. We then have $U_A(s^k) = U_A(\hat{s})$, and the utility obtained from using the strategy selected by Algorithm 2 at state s^k is no less than the utility obtained by following any other equilibrium strategy. Hence, there is no incentive to deviate from the equilibrium policy developed in Algorithm 2. □

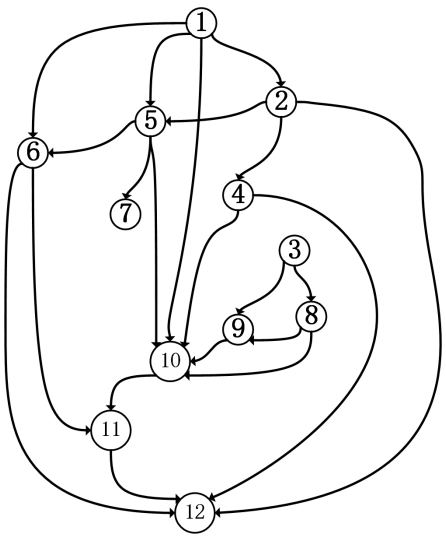


Figure 3.1: Provenance graph built based on real world log data from the RAIN system, showing the possible relationships between processes. The node 12 represents the ScreenGrab process, which is set to be the destination in the simulation. Node 1 and 3 are two possible origin processes, which the adversary can get access to at the first stage.

3.4 Simulation Study

To show Algorithm 1 is effective in deciding where to start tagging on real world log data, we have implemented Algorithm 1 based on a provenance graph of processes. The provenance graph is built from the log data generated by Refinable Attack Investigation System (RAIN) [65]. Specifically, in the simulation we consider the case where the ScreenGrab process is running in the system, which means the adversary may try to get access to the ScreenGrab to capture the screenshot of the victim's desktop and send to the attacker's server.

The provenance graph built from real log data is shown in Fig. 3.1. Node 12 represents the ScreenGrab process which the adversary wants to gain access to. From the log data, we obtain the fraction of flows that traverse each process in the graph, which we denote $Prob(s)$ (Table 3.1). We let $C_D(s) = cProb(s)$ for each node, where c is the tagging cost.

We choose a subset of all possible trap settings, because of limited space. We assume the

Table 3.1 Fraction of flows traversing nodes in provenance graph

Node #	1	2	3	4	5	6
Fraction	0.0308	0.0329	0.0175	0.0771	0.0164	0.0298
Node #	7	8	9	10	11	12
Fraction	0.0144	0.0010	0.0010	0.4121	0.0041	0.3628

security policy places traps on states with higher visiting probability. We consider the following three cases in the simulation.

Case 1: Parameters for the game are $\beta_A = 10$, $\beta_D = -10$, $\alpha_A = -10$, $\alpha_D = 10$ and $c = -400$. Results of case 1 are shown in Table 3.2. If the cost of tagging is high, it is not a good choice to

Table 3.2 Results for Case 1.

Trap states	Tagged states (probability p)	Utility of defender
4, 6, 10	5, 6, 8, 9 ($p = 0.5$)	-69.66
2, 4, 10	2, 8, 9 ($p = 0.5$)	-76.99
2, 4, 6, 10	1, 2, 5, 6, 8, 9 ($p = 0.5$)	-62.40
2, 4, 5, 6	2, 5, 6 ($p = 0.5$)	-95.83

start tagging from a frequently used state. For example, considering trap setting (4, 6, 10), since the probability of state 10 is high, a better way is to start tagging from state 5, 8 or 9 in order to detect the adversary while minimizing the performance overhead. When traps are placed on node

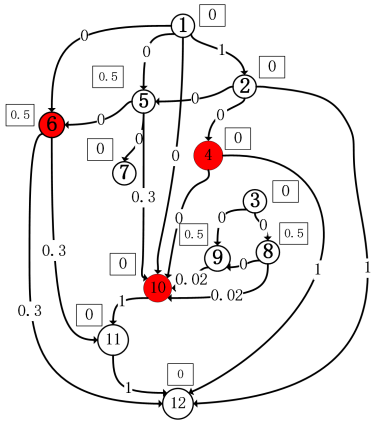


Figure 3.2: The output of Algorithm 1, when traps are on (4,6,10) in case 1. Red nodes are the states placed with traps. The number in a rectangular box is the tagging probability of the adjacent state, and the number on the line is the transition probability of the adversary.

4, 6 and 10, the transition probability of the adversary and tagging probability of the defender at equilibrium, are shown in Fig. 3.2. Different trap settings will result in different average utility of the defender, which is defined as $\frac{1}{|\mathcal{S}|}\sum_{i=1}^{|\mathcal{S}|} U_D(s_i)$. Another observation here is that more traps do not necessarily mean a higher average utility for the defender, which also depends on where the traps are placed. For instance, (2,4,6,10) is a good setting for the defender, resulting in the highest average utility. The setting (2,4,5,6), however, is the worst compared with the other 3 settings, even with more traps. From another perspective, it shows that our Algorithm can provide a quantitative way to measure how good the trap setting is, according to the average utility of the defender.

Case 2: Parameters for the game are $\beta_A = 500$, $\beta_D = -500$, $\alpha_A = -10$, $\alpha_D = 10$ and $c = -400$. Results of case 2 are shown in Table 3.3. In this case, we increase β_A and decrease β_D, corresponding to a strong benefit to the adversary from reaching the destination. From the simulation results, the best tagging points are on the trap states, which is different from Case 1. This is due to the fact that, in Case 2, the tagging cost is not significant, compared with β_D. The tagging probability is close to 1.

Case 3: Parameters for the game are $\beta_A = 500$, $\beta_D = -500$, $\alpha_A = -10$, $\alpha_D = 10$ and $c = -1000000$. Results of case 3 are shown in Table 3.4. For this case, the cost of tagging is high,

Table 3.3 Results for Case 2.

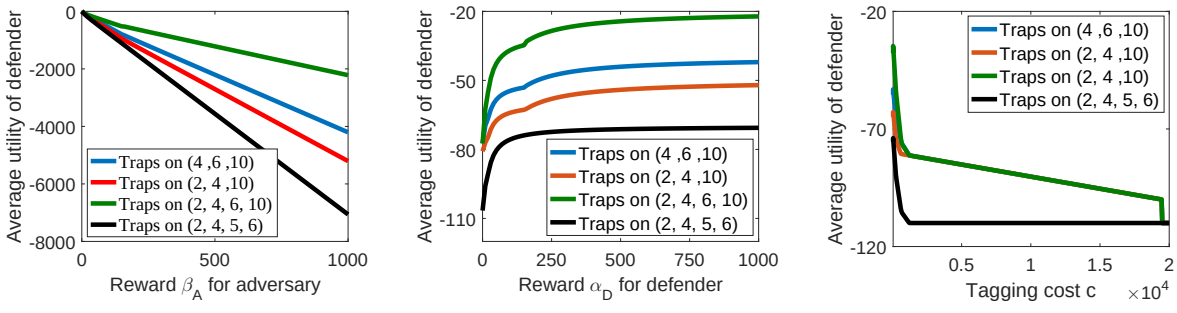
Trap states	Tagged states (probability p)	Utility of defender
4, 6, 10	4, 6, 10 (p=0.98)	-2203.5
2, 4, 10	2, 4, 10 (p=0.98)	-2704.7
2, 4, 6, 10	2, 4, 6, 10 (p= 0.98)	-1216.4
2, 4, 5, 6	2, 4, 5, 6 (p= 0.98)	-3561.3

Table 3.4 Results for case 3.

Trap states	Tagged states (probability p)	Utility of defender
4, 6, 10	Do not tag	-5500
2, 4, 10	Do not tag	-5500
2, 4, 6, 10	Do not tag	-5500
2, 4, 5, 6	Do not tag	-5500

so the defender does not tag anything.

To further investigate the impact of game parameters on the average utility of the defender, we perform more simulations for the same trap settings as in case 1. We let $\beta_A = -\beta_D$ and $\alpha_A = -\alpha_D$.The simulation can show how the average utility of the defender changes as a function of β_A, α_D or c, with all the other parameters fixed. Fig. 4.2(a) shows that the average utility of the defender is approximately linearly decreasing as β_A increases, which is in line with equation (6). As illustrated in Fig. 4.2(b), the average utility of the defender is increasing as α_D increases, but



(a) (b) (c)

Figure 3.3: (a) Average utility of a defender as a function of reward β_A for an adversary, where the α_D and c are fixed to 10 and -400, respectively. (b) Average utility of the defender as a function of reward α_D for the defender, where the α_A and c are fixed to 10 and -400, respectively. (c) Average utility of the defender as a function of a tagging cost c, where both α_A and β_D are fixed 10.

the increasing rate is slowing down. Moreover, we obtain the upper bound for the average utility of the defender, which is $\beta_D|\hat{\mathcal{S}}_1|$ as α_D goes to infinity. It should be noted that the sudden changes of slopes in Fig. 4.2(b) are due to the change of tagging states. Fig. 4.2(c) shows that by increasing the tagging cost, the average utility of the defender goes down. The extreme case is to set tagging cost c to be a very large number, then the utility of the defender reaches the lower bound, where the defender does not tag any flow.

3.5 Summary

APTs are stealthy, multi-stage attacks that are difficult to detect using standard defense mechanisms. Dynamic Information Flow Tracking (DIFT) has been proposed to detect APTs by tracking suspicious or unauthorized information flows, which are consistent across different stages of the attack. This chapter presented an analytical modeling framework for DIFT. We modeled the strategic interaction between an adversarial information flow and the DIFT-based defense as a multi-stage game. Each stage represented the attack progression through a particular system process, in which the attacker must determine which process to transition to at the next stage, while the defender

must decide whether to taint the traffic or not and incur additional performance overhead from tainting non-adversarial flows. We characterized the optimal adversary and defender strategies in two cases. In the first case, there is only a single destination, and we are able to derive a closed-form expression for the optimal strategies and resulting utilities. In the second case, there are multiple destinations with different adversary utilities for each destination, and we present an efficient algorithm for computing the optimal strategies. We demonstrate our model on a real-world dataset of the ScreenGrab attack obtained using the Refinable Attack Investigation (RAIN) framework.

Chapter 4

DIFT-APT GAMES FOR DETECTING MULTI-STAGE CYBER THREATS

4.1 Motivation

Our objective in this chapter is to develop a resource-efficient analytical model of DIFT to detect multi-stage APTs by an optimal tagging and trapping procedure. Adversarial interaction makes game theory a promising framework to characterize the trade-off between detection efficiency and cost of detection and develop an optimal DIFT defense, which is the contribution of this chapter. Each stage of the APT attack is a stage in our multi-stage game model and is characterized by a unique set of critical locations and critical infrastructures of the system, referred to as *destinations*. The contributions of this chapter are the following:

- We model the interaction of APTs and DIFT with the system as a two-player multi-stage nonzero-sum game with imperfect information structure. The adversary strategizes to reach a destination node and the defender strategizes to detect the APT in a resource-efficient manner. A solution to this game gives an optimal defense policy for DIFT to maximize the probability of APT detection while minimizing memory and performance overhead on the system.

- We develop algorithms to compute best responses of the players. The best response of the adversary is obtained by reducing it to a *shortest path problem* on a directed graph such that a shortest path corresponds to a path of maximum probability of reaching the final target. The best response of the defender, which is a subset of nodes, is approximated by the *submodularity* property of its payoff function.

- We analyze a special case of the problem where the attack is a single-stage attack. For this case, we characterize the set of *Nash equilibria* of the game. This characterization is obtained by

proving the equivalence of the dynamic game to a suitably defined bimatrix-game formulation.

- We provide a polynomial-time iterative algorithm to compute a local *correlated equilibrium* of the game for the multi-stage attack. Our algorithm provides locally optimal equilibrium strategies for both players by transforming the two-player game to an $(N(M+2)+|\Lambda|+1)$-player game, where N denotes the number of processes and objects in the system, M denotes the number of stages of the APT attack, and $|\Lambda|$ denotes the size of the set of security rules.

- We perform experimental analysis of our model on the real-world multi-stage attack data obtained using Refinable Attack INvestigation (RAIN) framework [64], [67] for a three day nation state attack.

4.2 DIFT-APT Game Formulation

In this section, we model a two-player multi-stage game between APTs and DIFT. We model the different stages of the game in such a way that each stage of the APT attack translates to a stage in the game.

4.2.1 System Model

We denote the adversarial player of the game by $\mathcal{P}_A$ and the defender player by $\mathcal{P}_D$. In the j^{th} stage of the attack, the objective of $\mathcal{P}_A$ is to evade detection and reach a destination node in stage j, i.e., $\mathcal{D}_j$. $\mathcal{P}_A$ can also abandon the attack at any stage by dropping out the flow. The objective of $\mathcal{P}_D$ is to detect $\mathcal{P}_A$ before $\mathcal{P}_A$ reaches a node in $\mathcal{D}_j$. In order to detect $\mathcal{P}_A$, $\mathcal{P}_D$ identifies a set of processes $\mathcal{Y} := \{y_1, \ldots, y_h\} \subseteq V_{\mathcal{G}}$ as the tag sources such that any information flow passing through a process $y_i \in \mathcal{Y}$ is tagged. $\mathcal{P}_D$ tracks the traversal of a tagged flow through the system and generates tag sinks denoted as $\mathcal{T} := \{t_1, \ldots, t_{h'}\} \subset V_{\mathcal{G}}$ using pre-specified security rules.

Let Λ be the set of security rules. We consider a security policy that is based on the terminal points of the flow, i.e., entry point of the flow and the location of the flow where analysis is performed. Therefore, $\Lambda : V_{\mathcal{G}} \times V_{\mathcal{G}} \rightarrow \{0, 1\}$, where 1 represents that the pair of terminal points

of the flow violates the security policy of the system and 0 otherwise. Here, $|\Lambda| \leqslant N^2$, since not all node pairs in $V_{\mathcal{G}}$ have a directed path between them. Hence the number of security rules that are relevant to a node is at most N. Without loss of generality, we assume that each node in $\mathcal{G}$ is associated with N security rules. As N is large, applying all N security rules at every tag sink may not be required. The security rules are pre-specified depending on the application running on the system and is known to the defender. In our game model, DIFT selects a subset of rules at every tag sink to perform security analysis.

4.2.2 State Space of the Game

Let $\lambda \subset V_{\mathcal{G}}$ denote the subset of nodes in the IFG that are susceptible (vulnerable) to attacks. In order to characterize the entry point of the attack by a unique node, we introduce a *pseudo-process* s_0 such that s_0 is connected to all the processes in the set λ. Let $\mathcal{S} := V_{\mathcal{G}} \cup \{s_0\}$, $E_\lambda := \{s_0\} \times \lambda$, and $\mathcal{E} := E_{\mathcal{G}} \cup E_\lambda$. Note that, s_0 is the root node of the modified graph and hence transitions are allowed *from* s_0 and no transition is allowed *into* s_0.

Now we define the state space of the game. Each decision point in the game is a state of the state space and is defined by the source of the flow in set λ, the stage of the attack, the current location in the IFG, s_i, along with its tag status, trap status, and the status of the N security rules applicable at s_i. We use s_i^j to denote the process s_i at the j^{th} stage of the attack. Then the state space of the game is denoted by

$$\bar{\mathcal{S}} := \{V_{\mathcal{G}} \times \{1, \ldots, M\} \times \lambda \times \{0,1\}^{2+N}\} \cup \{(s_0^1, \underbrace{0, \ldots, 0}_{2+N \text{ times}})\},$$

where $\bar{\mathcal{S}} = \{\bar{s}_1, \ldots, \bar{s}_T\}$ with $T = (2^{(2+N)} NM|\lambda|) + 1$. Here $\bar{s}_1 = (s_0^1, 0, \ldots, 0)$ is the state in $\bar{\mathcal{S}}$ corresponding to the pseudo-node s_0. The remaining states are given by $\bar{s}_r = (s_i^j, \lambda_r, k_r^1, \ldots, k_r^{(2+N)})$, for $r = 2, \ldots, T$, where $s_i \in V_{\mathcal{G}}$, $j \in \{1, \ldots, M\}$, $\lambda_r \in \{1, \ldots, |\lambda|\}$, and $k_r^1, \ldots, k_r^{2+N} \in \{0, 1\}$. Here, $k_r^1 = 1$ if the flow at s_i is tagged and $k_r^1 = 0$ otherwise. Similarly, $k_r^2 = 1$ if s_i is a tag sink and $k_r^2 = 0$ otherwise, and $k_r^3, \ldots, k_r^{2+N}$ denotes the selection of security rules (bit 1 denotes that a rule is selected and bit 0 otherwise).

Let $\mathcal{N}(s_i)$ denotes the set of out-neighbors of a node $s_i \in V_{\mathcal{G}}$ defined as $\mathcal{N}(s_i) := \{s_{i'} : (s_i, s_{i'}) \in \mathcal{E}\} \cup \{\phi\}$. Here ϕ corresponds to adversary dropping the flow. Consider two states $\bar{s}_r = (s_i^j, \lambda_r, k_r^1, \ldots, k_r^{(2+N)})$, $\bar{s}_{r'} = (s_{i'}^{j'}, \lambda_{r'}, k_{r'}^1, \ldots, k_{r'}^{(2+N)})$ in $\bar{\mathcal{S}}$. Then $\bar{s}_{r'}$ is an out-neighbor of $\bar{s}_r$ in the state space graph if one of the following cases hold: 1) $j = j'$ and $s_{i'} \in \mathcal{N}(s_i)$, and 2) $j' = j + 1$ and $s_i = s_{i'} \in \mathcal{D}_j$. Case 1) corresponds to transition in the same stage to an out-neighbor node or dropping out of the game and case 2) corresponds to transition at a destination from one stage to the next stage. Note that, in case 2) (i.e., $j' = j + 1$ and $s_i = s_{i'} \in \mathcal{D}_j$).

Tagging s_0 means tagging all sensitive flows which is not desirable on account of the performance overhead. Therefore, s_0 is neither a tag source nor a tag sink and it is always in stage 1 with origin at s_0 itself as denoted by state $\bar{s}_1$. We give the following definition for an adversarial flow in the state space $\bar{\mathcal{S}}$ originating at the state $(s_0^1, 0, \ldots, 0)$.

Definition 4.2.1. An information flow in $\bar{\mathcal{S}}$ that originates at state $(s_0^1, 0, \ldots, 0)$ and terminates at state $(s_i^j, \lambda_r, k_r^1, \ldots, k_r^{2+N})$ is said to satisfy the stage-constraint if the flow passes through some destinations in $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{j-1}$ in order.

Definition 4.2.1 holds in those applications in which the attacker must compromise the internal processes in a specific order in order to achieve the desired goal, e.g., accessing log-in ID, followed by password to access a bank account.

4.2.3 Actions of the Players

The players $\mathcal{P}_A$ and $\mathcal{P}_D$ have finite action sets over the state space $\bar{\mathcal{S}}$ denoted by sets $\mathcal{A}_A$ and $\mathcal{A}_D$, respectively. The action set of $\mathcal{P}_A$ is the set of all possible paths the attacker can traverse in an attack. Let Ω denote the set of all possible paths[1] in the graph $\hat{\mathcal{S}} = (V_{\hat{\mathcal{S}}}, E_{\hat{\mathcal{S}}})$ that originate at node s_0^1. Here $V_{\hat{\mathcal{S}}} := \{V_{\mathcal{G}} \times \{1, \ldots, M\}\} \cup \{s_0^1, \phi\}$ and $E_{\hat{\mathcal{S}}}$ consists of: i) $\{(s_0^1, s_i^1) : s_i \in \lambda\}$, ii) $\{(s_i^j, s_{i'}^j) : (s_i, s_{i'}) \in E_{\mathcal{G}}\}$, iii) $\{(s_i^j, s_i^{j+1}) : s_i \in \mathcal{D}_j\}$, and iv) $\{(s_i^j, \phi) : s_i \in V_{\mathcal{G}}, j \in \{1, \ldots, M\}\}$. Then the action set of $\mathcal{P}_A$ is Ω, i.e., $\mathcal{A}_A = \Omega$. Let $\bar{s}_r = (s_i^j, \lambda_r, k_r^1, \ldots, k_r^{(2+N)})$ be a state such that $s_i \in \mathcal{D}_j$. At this state, $\mathcal{A}_D = \emptyset$ and the next state of $\bar{s}_r$ is given by $(s_i^{j+1}, \lambda_r, k_r^1, \ldots, k_r^{(2+N)})$. This

[1]A path is a finite sequence of edges that join a sequence of vertices of a graph, without repetitions.

construction of $\bar{\mathcal{S}}$ captures the transitions of the attacker from stage j to stage $j+1$. Note that, $\mathcal{P}_A$ can also end the game by dropping the information flow at any point in time, i.e. when the end state of a path is a *null* state ϕ. For a state $(s_i^j, \lambda_r, k_r^1, \ldots, k_r^{(2+N)})$ in $\bar{\mathcal{S}}$, λ_r is decided by the process in λ to which the adversary transitions from s_0, i.e., the transition from $(s_0, 0, \ldots, 0)$ in the state space. Further, λ_r for a particular adversarial flow remains fixed for all states in $\bar{\mathcal{S}}$ that the flow traverses. As the tag propagation rules are pre-specified by the user, the action set of $\mathcal{P}_D$ includes selection of tag status of a flow, tag sinks, and security check rules. Hence the action set of the defender at s_i^j is a binary tuple, $(k_r^1, \ldots, k_r^{2+N})$, and $\mathcal{A}_D = \{0,1\}^{NM(2+N)}$. While the objective of $\mathcal{P}_A$ is to exploit the vulnerable processes λ of the system to successfully launch an attack, the objective of $\mathcal{P}_D$ is to select an optimal set of tagged nodes, say $\mathcal{Y}^\star \subset \mathcal{Y}$, and an optimal set of tag sinks, say $\mathcal{T}^\star \subset \mathcal{T}$, and a set of security rules such that any spurious information flow in the system is detected at some tag sink before reaching destination.

4.2.4 Information of the Game

Both adversary and the defender know the graph $\mathcal{G}$. At any state $\bar{s}_r$ in the game, the defender has the information about the tag source status of $\bar{s}_r$, the tag sink status of $\bar{s}_r$, and the set of security rules chosen at $\bar{s}_r$. However, the adversary is unaware of the tag source status, the tag sink status, and the security rules chosen at that state. On the other hand, while the adversary knows the stage of the attack, the defender does not know the stage of the attack and hence the unique set of destinations targeted by $\mathcal{P}_A$ in that particular stage. Thus, the players $\mathcal{P}_A$ and $\mathcal{P}_D$ have asymmetric knowledge resulting in an *imperfect information* game.

4.2.5 Strategies of the Players

A strategy is a rule that the player uses to select actions at every step of the game. We consider *mixed strategies* and hence there are probability distributions over the action sets $\mathcal{A}_A$ and $\mathcal{A}_D$. The defender's strategy, $\mathbf{p}_D$, consists of selecting the tag status, trap status, and security rules for flows passing through each state. The defender strategy at a state corresponding to process s_i, $\mathbf{p}_D(s_i)$, is

a tuple of length $2+N$, $(\mathbf{p}_D^1(s_i), \ldots, \mathbf{p}_D^{2+N}(s_i))$. Here, $\mathbf{p}_D^1(s_i)$ denotes the probability that the flow passing through s_i is tagged, $\mathbf{p}_D^2(s_i)$ the probability that s_i is a tag sink, and $(\mathbf{p}_D^3(s_i), \ldots, \mathbf{p}_D^{2+N}(s_i))$ the probability of selecting each rule in Λ corresponding to s_i. The pseudo-process s_0 has $\mathbf{p}_D^{i'}(s_0) = 0$ for $i' \in \{1, \ldots, 2+N\}$. Note that the defender strategy does not depend on the stage, as the defender is unaware of the stage of the attack. The adversary on the other hand knows the stage of the attack and hence the strategy of $\mathcal{P}_A$, i.e., the probability distribution on Ω, $\mathbf{p}_A : \Omega \to [0,1]^{|\mathcal{A}_A|}$, depends on the attack stage.

Taken together, the strategies of $\mathcal{P}_A$ and $\mathcal{P}_D$ are given by the vectors

$$\mathbf{p}_D = \{(\mathbf{p}_D^1(s_i), \ldots, \mathbf{p}_D^{2+N}(s_i)) : s_i \in \mathcal{S}\}$$

and

$$\mathbf{p}_A = \{\mathbf{p}_A(\omega) : \omega \in \Omega\},$$

respectively. Note that, $\mathbf{p}_A$ is a vector whose length equals the number of paths in Ω, while $\mathbf{p}_D$ is a vector of length $|\mathcal{S}|$ with each entry of length $(2+N)$. Notice that $\mathbf{p}_A$ is defined in such a way that a flow that originate at $(s_0^1, 0, \ldots, 0)$ in the state space $\bar{\mathcal{S}}$ reaches a state $(s_i^j, \lambda_r, k_r^1, \ldots, k_r^{2+N})$, for some $\lambda_r \in \{1, \ldots, |\lambda|\}$ and for some $k_r^1, \ldots, k_r^{2+N} \in \{0,1\}$, after passing through some destinations of stages $1, \ldots, j-1$. By this definition of state space and strategies of the game, all information flows in $\bar{\mathcal{S}}$ satisfy the stage-constraints, given in Definition 4.2.1, and can affect the performance of the system and even result in system breakdown, if malicious.

4.2.6 Payoffs to the Players

Now we define the payoffs of players $\mathcal{P}_A$ and $\mathcal{P}_D$, denoted by U_A and U_D, respectively. The payoffs for players include penalties and rewards at every stage of the attack. U_A consists of: (i) reward $\beta_j^A > 0$ for adversary successfully reaching a destination in the j^{th} stage satisfying the stage-constraints, and (ii) cost $\alpha^A < 0$ if the adversary is detected by the defender. Similarly, U_D consists of: (a) memory cost $\mathcal{C}_D(s_i) < 0$ for tagging node $s_i \in V_{\mathcal{G}}$, (b) memory cost $\mathcal{W}_D(s_i) < 0$ for setting tag sink at node $s_i \in V_{\mathcal{G}}$, (c) cost γ_i, for $i \in \{1, \ldots, N\}$, for selecting the i^{th} security check rule at a tag sink, (d) cost $\beta_j^D < 0$ if the adversary reaches a destination in the j^{th} stage satisfying

the stage-constraint, and (e) reward $\alpha^D > 0$ for detecting the adversary. We assume that the cost of tagging a node and the cost of setting tag sink at a node, $\mathcal{C}_D(s_i)$ and $\mathcal{W}_D(s_i)$, respectively, are independent of the attack stage. However, $\mathcal{C}_D(s_i)$ and $\mathcal{W}_D(s_i)$ depends on the average traffic at process s_i and hence $\mathcal{C}_D(s_i) := c_1\, B(s_i)$ and $\mathcal{W}_D(s_i) := c_2\, B(s_i)$. Here, $c_1 \in \mathbb{R}_-$ is a fixed tagging cost and $c_2 \in \mathbb{R}_-$ is a fixed cost for setting tag sink, where $\mathbb{R}_-$ is the set of negative real numbers, and $B(s_i)$ denotes the average traffic at node s_i. At a state $\bar{s}_r = (s_i^j, \lambda_r, k_r^1, \ldots, k_r^{(2+N)})$, the costs $\mathcal{C}_D(s_i)$ and $\mathcal{W}_D(s_i)$ are incurred if the corresponding bit denoting the tag status and trap status, i.e., k_r^1, k_r^2, respectively, are 1.

Recall that, the origin of any adversarial information flow in the state space $\bar{\mathcal{S}}$ is $(s_0^1, 0, \ldots, 0)$. For a flow originating at state $(s_0^1, 0, \ldots, 0)$ in $\bar{\mathcal{S}}$, let $p_T(j)$ denotes the probability that the flow will get detected at stage j and $p_R(j)$ denotes the probability that the flow will reach some destination in set $\mathcal{D}_j$. The values of $p_T(j)$ and $p_R(j)$ depend on the tag status, the tag sink status, and the set of security rules selected. For a given strategy, $(\mathbf{p}_D, \mathbf{p}_A)$, the payoffs U_D and U_A are given by,

$$\begin{aligned} U_D(\mathbf{p}_D, \mathbf{p}_A) &= \sum_{s_i \in V_{\mathcal{G}}} \Big(\mathbf{p}_D^1(s_i)\, \mathcal{C}_D(s_i) + \mathbf{p}_D^2(s_i)\, \mathcal{W}_D(s_i) + \\ &\qquad \sum_{g=1}^{N} \mathbf{p}_D^{2+g}(s_i)\, \gamma_r \Big) + \sum_{j=1}^{M} \Big(p_T(j)\alpha^D + p_R(j)\beta_j^D \Big), \qquad (4.1) \\ U_A(\mathbf{p}_D, \mathbf{p}_A) &= \sum_{j=1}^{M} \Big(p_T(j)\alpha^A + p_R(j)\beta_j^A \Big). \qquad (4.2) \end{aligned}$$

Note that, if the adversarial flow is detected in stage j, then $p_T(j') = 0$ for $j' > j$ and $j' \in \{1, \ldots, M\}$.

Assumption 4.2.2. If the game parameters, $\alpha^D, \alpha^A, \beta^D, \beta^A, \mathcal{C}_D, \mathcal{W}_D$, are such that the cost of tagging a flow and performing security analysis across a path is high enough that there could be paths that may not be worth the tagging cost, then the adversary always achieves its goal irrespective of the defender's actions. To avoid this trivial case, we assume that the game parameters satisfy the condition that at equilibrium there exists a defender strategy with nonzero probability of detection.

4.2.7 Preliminary Analysis of the Model

In this subsection, we perform an initial analysis of our model. A multi-stage attack consisting of M stages belongs to one of the following $M + 2$ scenarios.

1) The adversary drops out of the game before reaching some destination in $\mathcal{D}_1$.

2) The adversary reaches some destination in each of $\mathcal{D}_1, \ldots, \mathcal{D}_j$ and then drops out of the game, for $j = 1, \ldots, M-1$ ($M-1$ possibilities).

3) The adversary reaches some destination in each of $\mathcal{D}_1, \ldots, \mathcal{D}_M$.

4) The defender detects the adversary at some stage.

The payoff of the game is different for each of the cases listed above. In scenario 1), $\mathcal{P}_A$ and $\mathcal{P}_D$ incur zero payoff. In scenario 2), $\mathcal{P}_A$ earns rewards for reaching stages $1, \ldots, j$, respectively, $\mathcal{P}_D$ incurs the penalty for not detecting the adversary at stages $1, \ldots, j$, respectively, and the game terminates. In scenario 3), $\mathcal{P}_A$ earns rewards for reaching destinations in all stages and wins the game and $\mathcal{P}_D$ incurs a total penalty for not detecting the adversary at all the stages. In scenario 4), $\mathcal{P}_A$ incurs the penalty for getting detected and $\mathcal{P}_D$ earns the reward for detecting the adversary and wins the game.

For calculating the payoffs of $\mathcal{P}_D$ and $\mathcal{P}_A$ at a decision point in the game (i.e., at a state in $\bar{\mathcal{S}}$), for given player strategies $(\mathbf{p}_D, \mathbf{p}_A)$, we define payoffs at a state $L^A_{(\mathbf{p}_D, \mathbf{p}_A)} : \bar{\mathcal{S}} \rightarrow \mathbb{R}$ and $L^D_{(\mathbf{p}_D, \mathbf{p}_A)} : \bar{\mathcal{S}} \rightarrow \mathbb{R}$ for the adversary and defender, respectively, at every state in the state space $\bar{\mathcal{S}}$. Let the current state of the game be $(s_i^{j'}, \lambda_r, k_r^1, \ldots, k_r^{2+N})$, where $\lambda_r \in \{1, \ldots, |\lambda|\}$ and $k_r^1, \ldots, k_r^{2+N} \in \{0, 1\}$, and $q(s_i^{j'})$ denote the probability that the next state of the game is ϕ. Further, consider the set of paths, $\Omega_j \subset \Omega$, that originate at s_0^1, reaches a destination node in $\mathcal{D}_j$, and then drops out before reaching a destination in $\mathcal{D}_{j+1}$, without getting detected by the defender. Let $P_{R,j}(s_i^{j'}, \lambda_r, k_r^1, \ldots, k_r^{2+N})$ denote the probability that the adversary reaches some node in $\mathcal{D}_j$ and drops outs before reaching $\mathcal{D}_{j+1}$, provided the current state of the game is $(s_i^{j'}, \lambda_r, k_r^1, \ldots, k_r^{2+N})$. Also let $P_T(s_i^{j'}, k_r^1, \ldots, k_r^{2+N})$ denote the probability that information

flow is detected by the defender when the current state is $(s_i^{j'}, \lambda_r, k_r^1, \ldots, k_r^{2+N})$. To characterize the payoffs of the players at a state in $\bar{\mathcal{S}}$, we now introduce a few notations. For notational brevity, let us denote $k_r^1, \ldots, k_r^{2+N}$ by $\bar{k}_r$, for $r = 1, \ldots, N$. For state $(s_i^{j'}, \lambda_r, \bar{k}_r)$ and $t \in \{1, \ldots, M\}$,

$$Q_t(s_i^{j'}) := \sum_{\substack{\omega \in \Omega_j \\ s_\ell \in \mathcal{N}(s_i) \\ k_\ell^g \in \{0,1\} \\ g \in \{1,\ldots,2+N\}}} \mathbf{p}_{A}(\omega) \Big[\prod_{g=1}^{2+N} \Big(\mathbf{p}_D^g(s_\ell) \Big)^{k_\ell^g} \Big(1 - \mathbf{p}_{D}^{g}(s_\ell) \Big)^{(1-k_\ell^g)} \Big] P_{R,t}(s_\ell^t, \lambda_r, \bar{k}_r),$$

$$\overline{Q}_t(s_i^{j'}) := \sum_{\substack{\omega \in \Omega_j \\ s_\ell \in \mathcal{N}(s_i) \\ k_\ell^g \in \{0,1\} \\ g \in \{1,\ldots,2+N\}}} \mathbf{p}_{A}(\omega) \Big[\prod_{g=1}^{2+N} \Big(\mathbf{p}_D^g(s_\ell) \Big)^{k_\ell^g} \Big(1 - \mathbf{p}_{D}^{g}(s_\ell) \Big)^{(1-k_\ell^g)} \Big] P_T(s_\ell^t, \lambda_r, \bar{k}_r).$$

Then,

$$P_{R,j}(s_i^{j'}, \lambda_r, \bar{k}_r) = \begin{cases} 0, & k_r^1 = \cdots = k_r^{2+N} = 1 \\ q(s_i^{j'}) + Q_{j'+1}(s_i^{j'}), & s_i \in \mathcal{D}_j,\ j' = j \\ 0, & s_i \in \mathcal{D}_{j'}, j' = j+1 \\ 0, & j' > j+1 \\ Q_{j'}(s_i^{j'}), & j' \leq j \\ q(s_i^{j'}) + Q_{j'}(s_i^{j'}), & j' = j+1 \end{cases}$$

$$P_T(s_i^{j'}, \lambda_r, \bar{k}_r) = \begin{cases} 1, & k_r^1 = \cdots = k_r^{2+N} = 1 \\ 0, & j' = M, s_i \in \mathcal{D}_M \\ \overline{Q}_{j'}(s_i^{j'}), & \text{otherwise.} \end{cases}$$

Using the definitions of $P_{R,j}(\cdot)$ and $P_T(\cdot)$ at a state in $\bar{\mathcal{S}}$, the payoffs of $\mathcal{P}_D$ and $\mathcal{P}_A$ at a state $(s_i^{j'}, \lambda_r, k_r^1, \ldots, k_r^{2+N})$ are given by Eqs. (4.3) and (4.4), respectively.

$$L^{D}_{(\mathbf{p}_D,\mathbf{p}_A)}(s_i^{j'},\lambda_r,\bar{k}_r) = \sum_{s_b\in V_{\mathcal{G}}} \Big(p^1_{F,b}(s_i^{j'},\lambda_r,\bar{k}_r)\,\mathcal{C}_D(s_b) + p^2_{F,b}(s_i^{j'},\lambda_r,\bar{k}_r)\,\mathcal{W}_D(s_b) + \sum_{g=1}^{N} p^{2+g}_{F,b}(s_i^{j'},\lambda_r,\bar{k}_r)\,\gamma_g \Big) + \sum_{j=1}^{M} \Big(p_{R,j}(s_i^{j'},\lambda_r,\bar{k}_r)(\sum_{v=1}^{j}\beta_v^D) + P_T(s_i^{j'},\lambda_r,\bar{k}_r)\alpha^D \Big), \quad (4.3)$$

$$L^{A}_{(\mathbf{p}_D,\mathbf{p}_A)}(s_i^{j'},\lambda_r,\bar{k}_r) = \sum_{j=1}^{M} \Big(p_{R,j}(s_i^{j'},\lambda_r,\bar{k}_r)(\sum_{v=1}^{j}\beta_v^A) + P_T(s_i^{j'},\lambda_r,\bar{k}_r)\alpha^A \Big). \quad (4.4)$$

In Eqs. (4.3) and (4.4), $p^1_{F,b}(s_i^{j'},\lambda_r,\bar{k}_r)$ denotes the probability that flow passing through $s_b \in V_{\mathcal{G}}$ is tagged if the current state is $(s_i^{j'},\lambda_r,\bar{k}_r)$ and $p^2_{F,b}(s_i^{j'},\lambda_r,\bar{k}_r)$ denotes the probability that node $s_b \in V_{\mathcal{G}}$ is a tag sink in a flow whose current state is $(s_i^{j'},\lambda_r,\bar{k}_r)$. Similarly, $p^{2+g}_{F,b}(s_i^{j'},\lambda_r,\bar{k}_r)$ denotes the probability that the r^{th} security rule is selected for inspecting authenticity of a flow whose current state is $(s_i^{j'},\lambda_r,\bar{k}_r)$. Eqs. (4.3) and (4.4) give a system of $2^{(2+N)}NM|\lambda|+1$ linear equations each for the payoff functions $L^{D}_{(\mathbf{p}_D,\mathbf{p}_A)}$ and $L^{A}_{(\mathbf{p}_D,\mathbf{p}_A)}$, where $L^{D}_{(\mathbf{p}_D,\mathbf{p}_A)}(b)$, $L^{A}_{(\mathbf{p}_A,\mathbf{p}_A)}(b)$ denote the payoffs at the b^{th} state in $\bar{\mathcal{S}}$. Lemma 4.2.3 relates payoffs of the game $U_D(\mathbf{p}_D,\mathbf{p}_A), U_A(\mathbf{p}_D,\mathbf{p}_A)$ with payoffs at a state $L^{D}_{(\mathbf{p}_D,\mathbf{p}_A)}, L^{A}_{(\mathbf{p}_A,\mathbf{p}_A)}$, respectively. We use Lemma 4.2.3 to compute a local correlated equilibrium of the game in Section 4.3.3 (Algorithm 4, Step 15).

Lemma 4.2.3. Consider the defender and adversary strategies $\mathbf{p}_D$ and $\mathbf{p}_A$, respectively. Then, the following hold: (i) $U_A(\mathbf{p}_D,\mathbf{p}_A) = L^{A}_{(\mathbf{p}_D,\mathbf{p}_A)}(s_0^1,0,\ldots,0)$, and (ii) $U_D(\mathbf{p}_D,\mathbf{p}_A) = L^{D}_{(\mathbf{p}_D,\mathbf{p}_A)}(s_0^1,0,\ldots,0)$.

Proof. **(i)**: By definition we get, $L^{A}_{(\mathbf{p}_D,\mathbf{p}_A)}(s_0^1,0,\ldots,0) = \sum_{j=1}^{M} \Big(P_{R,j}(s_0^1,0,\ldots,0)(\sum_{v=1}^{j}\beta_v^A) + P_T(s_0^1,0,\ldots,0)\alpha^A \Big)$. Here,

$$\sum_{j=1}^{M} P_{R,j}(s_0^1,0,\ldots,0)(\sum_{v=1}^{j}\beta_v^A) = \beta_1^A \sum_{j=1}^{M} P_{R,j}(s_0^1,0,\ldots,0) + \beta_2^A \sum_{j=2}^{M} P_{R,j}(s_0^1,0,\ldots,0) + \ldots + \beta_M^A\, P_{R,M}(s_0^1,0,\ldots,0), \quad (4.5)$$

Where, $\sum_{j=1}^{M} p_{R,j}(s_0^1,0,\ldots,0)$ is the total probability that a flow originating at $(s_0^1,0,\ldots,0)$ reach some destination in $\mathcal{D}_1$. Similarly, $\sum_{j=2}^{M} p_{R,j}(s_0^1,0,\ldots,0)$ is the total probability that a

flow originating at $(s_0^1, 0, \ldots, 0)$ reach some destination in $\mathcal{D}_2$. Thus

Thus, $\sum_{j=1}^{M} P_{R,j}(s_0^1, 0, \ldots, 0) = p_R(1)$. Similarly,

$$\sum_{j=2}^{M} P_{R,j}(s_0^1, 0, \ldots, 0) = p_R(2), \ldots, P_{R,M}(s_0^1, 0, \ldots, 0) = p_R(M). \quad (4.6)$$

From Eqs. (4.5) and (4.6), we get

$$\sum_{j=1}^{M} \Big(P_{R,j}(s_0^1, 0, \ldots, 0)(\sum_{v=1}^{j} \beta_v^A) \Big) = \sum_{j=1}^{M} p_R(j)\beta_j^A. \quad (4.7)$$

Since $P_T(s_0^1, 0, \ldots, 0) = \sum_{j=1}^{M} p_T(j)$,

$$P_T(s_0^1, 0, \ldots, 0)\alpha^A = \sum_{j=1}^{M} p_T(j)\alpha^A. \quad (4.8)$$

From Eqs. (4.7) and (4.8), we get $L^A_{(\mathbf{p}_D, \mathbf{p}_A)}(s_0^1, 0, \ldots, 0) = \sum_{j=1}^{M} \Big(p_R(j)\beta_j^A + p_T(j)\alpha^A \Big) = U_A(\mathbf{p}_D, \mathbf{p}_A)$.

(ii): Notice that $p^1_{F,i}(s_0^1, 0, \ldots, 0)$ is the probability that the process s_i is a tag source in a flow originating at $(s_0^1, 0, \ldots, 0)$. Thus $p^1_{F,i}(s_0^1, 0, \ldots, 0) = \mathbf{p}_D^1(s_i)$. Similarly, we get $p^2_{F,i}(s_0^1, 0, \ldots, 0) = \mathbf{p}_D^2(s_i)$ and $p^{2+g}_{F,i}(s_0^1, 0, \ldots, 0) = \mathbf{p}_D^{2+g}(s_i)$, for $g = 1, \ldots, N$. This along with Eqs. (4.7) and (4.8) implies that $L^D_{(\mathbf{p}_D, \mathbf{p}_A)}(s_0^1, 0, \ldots, 0) = \sum_{s_i \in V_\mathcal{G}} \Big(\mathbf{p}_D^1(s_i)\mathcal{C}_D(s_i) + \mathbf{p}_D^2(s_i)\mathcal{W}_D(s_i) + \sum_{g=1}^{N} \mathbf{p}_D^{2+g}(s_i)\gamma_r \Big) + \sum_{j=1}^{M} \Big(p_R(j)\beta_j^D + p_T(j)\alpha^D \Big) = U_D(\mathbf{p}_D, \mathbf{p}_A)$. This completes the proof of (i) and (ii). □

4.3 Solution to DIFT-APT Game

This section presents an overview of the notions of equilibrium considered in this work. We first describe the concept of a player's best response to a given mixed policy of an opponent.

Definition 4.3.1. Let $\mathbf{p}_A : \Omega \to [0,1]^{|\Omega|}$ denote an adversary strategy (probability of selecting paths) and $\mathbf{p}_D : \mathcal{S} \to [0,1]^{(2+N)|\mathcal{S}|}$ denote a defender strategy (probabilities of tagging, tag sink selection, and security rule selection at every node in the graph). The set of best responses of the defender given by

$$\text{BR}(\mathbf{p}_A) = \arg\max_{\mathbf{p}_D} \{U^D(\mathbf{p}_D, \mathbf{p}_A) : \mathbf{p}_D \in [0,1]^{(2+N)|\mathcal{S}|}\}.$$

Similarly, the best responses of the adversary are given by

$$\text{BR}(\mathbf{p}_D) = \arg\max_{\mathbf{p}_A} \{U^A(\mathbf{p}_D, \mathbf{p}_A) : \mathbf{p}_A \in [0,1]^{|\Omega|}\}.$$

Intuitively, the best responses of the defender are the set of tagging strategies, the set of tag sink selection strategies, and the set of security rule selection strategies that jointly maximize the defender's payoff for a given adversary strategy. At the same time, the best responses of the adversary are the sets of probabilities of paths that maximize the adversary's payoff for a given defender (tagging, tag sink selection, and security rule selection) strategy. A mixed policy profile is a *Nash equilibrium* (NE) if the mixed policy of each player is a best response to the fixed mixed policy of the rest of the players. Formal definition of Nash equilibrium is as follows:

Definition 4.3.2. A pair of mixed policies $(\mathbf{p}_D, \mathbf{p}_A)$ is a *Nash equilibrium* if

$$\mathbf{p}_D \in \text{BR}(\mathbf{p}_A) \text{ and } \quad \mathbf{p}_A \in \text{BR}(\mathbf{p}_D).$$

A Nash equilibrium (NE) captures the notion of a stable solution as it occurs when neither player can improve its payoff by unilaterally changing its strategy. A pure strategy NE for the APT vs. DIFT game corresponds to the adversary deterministically choosing a path from the entry point to a destination node. However, in that case the defender can always improve the payoff by performing security analysis at only one node in that path with probability 1. Hence there exists no pure strategy NE for the game. Nash's result in [95] that proved the existence of an NE for a finite game with mixed strategy guarantees the existence of NE for the game we consider in this chapter. While there exists an NE for games with rational, noncooperative players, it is PPAD-complete to compute it in general [39], especially for nonzero-sum dynamic games of the type considered in this chapter. Also note that, for the game considered in this chapter, the payoff functions for the players are nonlinear in the probabilities. A weaker solution concept which is a relaxation of the Nash equilibrium is the *correlated equilibrium* defined as follows.

Definition 4.3.3. Let P denote a joint probability distribution over the set of defender and adversary strategies. Then P is a *correlated equilibrium* if for all strategies $\mathbf{p}_A, \mathbf{p}'_A$ and $\mathbf{p}_D, \mathbf{p}'_D$, conditioned

on that the strategy drawn from P is $(\mathbf{p}_D, \mathbf{p}_A)$,

$$\begin{aligned} U^D(\mathbf{p}_D, \mathbf{p}_A) &\geq U^D(\mathbf{p}'_D, \mathbf{p}_A) \\ U^A(\mathbf{p}_D, \mathbf{p}_A) &\geq U^A(\mathbf{p}_D, \mathbf{p}'_A) \end{aligned}$$

We next consider a simpler version of the correlated equilibrium that models the local policies at each process.

Definition 4.3.4. Let P denote a joint probability distribution over the set of defender and adversary actions. The distribution P is a *local correlated equilibrium* if for all states $s_i \in \mathcal{S}$, $j \in \{1, \ldots, M\}$, and strategies $\hat{\mathbf{p}}_D(s_i)$ and $\hat{\mathbf{p}}_A(\omega)$, conditioned on that the strategy drawn from P is $(\mathbf{p}_D, \mathbf{p}_A)$, we have

$$\begin{aligned} U^D(\mathbf{p}_D, \mathbf{p}_A) &\geq U^D(\mathbf{p}'_D, \mathbf{p}_A) \\ U^A(\mathbf{p}_D, \mathbf{p}_A) &\geq U^A(\mathbf{p}_D, \mathbf{p}'_A) \end{aligned}$$

where $\mathbf{p}'_D$ denotes a strategy with $\mathbf{p}'^x_D(s_i) = \hat{\mathbf{p}}^x_D(s_i)$, for some $x \in \{1, \ldots, 2+N\}$, $\mathbf{p}'^y_D(s_i) = \mathbf{p}^y_D(s_i)$ for $y \in \{1, \ldots, 2+N\}, y \neq x$, and $\mathbf{p}'_D(s_{i'}) = \mathbf{p}_D(s_{i'})$ for $i \neq i'$, and $\mathbf{p}'_A$ denotes a strategy with $\mathbf{p}'_A(\omega) = \hat{\mathbf{p}}_A(\omega)$ and $\mathbf{p}'_A(\omega') = \mathbf{p}_A(\omega')$ for $\omega \neq \omega'$.

4.3.1 Best Response of the Players

In this section, we calculate the best responses of players, $\mathcal{P}_A$ and $\mathcal{P}_D$.

Best Response for the Adversary

The best response of the adversary to a given defender strategy is described here. Firstly, we present the following preliminary lemma.

Lemma 4.3.5. Consider a defender policy $\mathbf{p}_D$. For each destination $d^j_b \in \mathcal{D}_j$, let $\Omega_{d^j_b}$ denote the set of paths in $\hat{\mathcal{S}}$ that originate at s^1_0 and terminate at some state that corresponds to node d^j_b. For any path ω, let $p(\omega)$ denote the probability that a flow reaches the destination without getting

detected by the defender. Finally, for every d_b^j, choose a path $\omega^*_{d_b^j} \in \arg\max\{p(\omega) : \omega \in \Omega_{d_b^j}\}$. Let $\omega^* \in \arg\max\{p(\omega_{d_b^j}) : d_b^j \in \mathcal{D}_j, j = 1, \ldots, M\}$. Finally, define the policy $\mathbf{p}_A^*$ by

$$\mathbf{p}_A^*(s_i^j, s_{i'}^{j'}) = \begin{cases} 1, & (s_i^j, s_{i'}^{j'}) \in \omega^* \\ 0, & \text{else} \end{cases}$$

Then, $\omega^* \in \text{BR}(\mathbf{p}_D)$.

Proof. Let $\mathbf{p}_A$ be any adversary policy, and let Ω denote the set of all possible paths in $\hat{\mathcal{S}}$ that are chosen by the adversary with nonzero probability such that the termination of the path is at some destination in $\mathcal{D} = \cup_{j=1}^M \mathcal{D}_j$. The payoff of the adversary can be written as

$$\begin{aligned} U^A(\mathbf{p}_D, \mathbf{p}_A) &= \sum_{\omega \in \Omega} \pi(\omega)(p(\omega)\beta^A_{j(\omega)} + (1 - p(\omega))\alpha^A) \\ &= \sum_{j=1}^{M} \sum_{d_b^j \in \mathcal{D}_j} \sum_{\omega \in \Omega_{d_b^j}} \pi(\omega)(p(\omega)\beta_j^A + (1 - p(\omega))\alpha^A, \end{aligned}$$

where $j(\omega)$ is equal to the stage where the path terminates and $\pi(\omega)$ is the probability that the path is chosen under this policy. The payoff $U^A(\mathbf{p}_D, \mathbf{p}_A)$ is bounded above by the path that maximizes $p(\omega)(\beta_j^A - \alpha^A)$, which is exactly the path ω^*. □

Using Lemma 4.3.5, we present the following approach to select a best response to the adversary for a given defender policy. For each destination in $\bigcup_{j=1}^M \mathcal{D}_j$, we first choose a path ω to that destination such that the probability of reaching that destination, $p(\omega)$, is maximized while traversing destinations of all intermediate stages. From those paths, we then select a path that maximizes $p(\omega)(\beta_j^A - \alpha^A)$.

Proposition 4.3.6. The path ω^* returned by a shortest path algorithm on the state space graph with edge weights of each incoming edge to states that correspond to node s_i equal to $-\log\left(1 - \mathbf{p}_D^1(s_i)\mathbf{p}_D^2(s_i)\prod_{r=3}^{2+N}\mathbf{p}_D^r(s_i)\right)$ is a best response to the defender strategy $\mathbf{p}_D$.

Proof. Consider a path $\omega \in \Omega_{d_b^j}$, i.e., a path that originates at s_0^1 and terminates at some state that correspond to node d_b^j. Let the first node that belongs to the vulnerable set λ, through which ω

traverses be denoted by λ_ω. For all nodes of $\mathcal{G}$ that lie in ω, i.e., $s_i \in \omega$, let $\Lambda^\omega(s_i)$ denote the set of indices of the security rules in Λ that are based on s_i and λ_ω.

By Lemma 4.3.5, it suffices to show that a shortest path in the state space with a suitably defined weight function will return a path in $\hat{\mathcal{S}}$ with the maximum probability of reaching some state in $\bar{\mathcal{S}}$ corresponding to node d_b^j of $\mathcal{G}$ without getting detected by the adversary. For any path $\omega \in \Omega_{d_b^j}$, the probability that the flow reaches d_b^j without getting detected by the adversary is equal to $\prod_{s_i \in \omega} \left(1 - \mathbf{p}_D^1(s_i)\mathbf{p}_D^2(s_i) \prod_{r=3}^{2+N} \mathbf{p}_D^r(s_i)\right)$.

$$\begin{aligned}
&\arg\max_{\omega \in \Omega_{d_b^j}} \prod_{s_i \in \omega} \left(1 - \mathbf{p}_D^1(s_i)\mathbf{p}_D^2(s_i) \prod_{r=3}^{2+N} \mathbf{p}_D^r(s_i)\right) \\
&= \arg\max_{\omega \in \Omega_{d_b^j}} \sum_{s_i \in \omega} \log \left(1 - \mathbf{p}_D^1(s_i)\mathbf{p}_D^2(s_i) \prod_{r=3}^{2+N} \mathbf{p}_D^r(s_i)\right) \\
&= \arg\min_{\omega \in \Omega_{d_b^j}} -\sum_{s_i \in \omega} \log \left(1 - \mathbf{p}_D^1(s_i)\mathbf{p}_D^2(s_i) \prod_{r=3}^{2+N} \mathbf{p}_D^r(s_i)\right)
\end{aligned}$$

The problem of finding best response to the adversary is equivalent to finding the shortest path from s_0^1 to d_b^j in a graph where the edge weights, which are non-negative as

$$\left(1 - \mathbf{p}_D^1(s_i)\mathbf{p}_D^2(s_i) \prod_{r=3}^{2+N} \mathbf{p}_D^r(s_i)\right) \leqslant 1,$$

are equal to $-\log \left(1 - \mathbf{p}_D^1(s_i)\mathbf{p}_D^2(s_i) \prod_{r=3}^{2+N} \mathbf{p}_D^r(s_i)\right)$ for each edge incoming to $s_i^{j'}$, for $j' \in \{1, \ldots, M\}$. □

Best Response for the Defender

We now present an approach for approximating the best response of the defender. In this approach, the set of possible responses at $s_i \in V_\mathcal{G}$ is discretized. Define

$$V_r = \{s_i^{z_r} : s_i \in V_\mathcal{G}, z_r = 1, \ldots, Z_r\}$$

for integers $Z_r > 0$, where $r = 1, \ldots, 2 + N$. For any $V_r' \subseteq V_r$, $r = 1, \ldots, 2 + N$, define

$$\mathbf{p}_D(s_i; V_r') \;=\; \frac{1}{Z_r} |\{s_i^{z_r} : z_r = 1, \ldots, Z_r\} \cap V_r'|, \tag{4.9}$$

and define $\mathbf{p}_D(V_1')$ to be the resulting vector of probabilities for tag selection, $\mathbf{p}_D(V_2')$ to be the resulting vector of probabilities for tag sink selection, and $\mathbf{p}_D(V_3'), \ldots, \mathbf{p}_D(V_{2+N}')$ to be the resulting vectors of probabilities for security rule selection. Then $\mathbf{p}_D(V') = \{\mathbf{p}_D(V_r')\}_{r=1}^{2+N}$ is the resulting vector of defender strategy, where $V' = \{V_1', \ldots, V_{2+N}'\}$. For a given adversary strategy, say $\mathbf{p}_A$, let $f(V') = U^D(\mathbf{p}_D(V'), \mathbf{p}_A)$.

Proposition 4.3.7. The function $f(V')$ is submodular as a function of V', that is, for any V_r', V_r'' with $V_r' \subseteq V_r''$ and any $s_i^{z_r} \notin V_r''$, for $r \in \{1, \ldots, 2+N\}$,

$$f\Big(\Big\{V_r' \cup \{s_i^{z_r}\}\Big\}_{r=1}^{2+N}\Big) - f(V') \geq f\Big(\Big\{V_r'' \cup \{s_i^{z_r}\}\Big\}_{r=1}^{2+N}\Big) - f(V'').$$

Proof. Consider $U^D(\mathbf{p}_D, \mathbf{p}_A)$ as defined in Eq. (6.2). The first and second terms of $U^D(\mathbf{p}_D, \mathbf{p}_A)$ are equal to

$$\sum_{s_i \in V_{\mathcal{G}}} \frac{\mathcal{C}^D(s_i)}{Z_1} |\{s_i^{z_1} : z_1 = 1, \ldots, Z_1\} \cap V_1'| \text{ and}$$

$$\sum_{s_i \in V_{\mathcal{G}}} \frac{\mathcal{W}^D(s_i)}{Z_2} |\{s_i^{z_2} : z_2 = 1, \ldots, Z_2\} \cap V_2'|,$$

respectively, both of which are modular as a function of V'. The third term of $U^D(\mathbf{p}_D, \mathbf{p}_A)$ equals

$$\sum_{s_i \in V_{\mathcal{G}}} \sum_{r=1}^{N} \frac{\gamma_r}{Z_{2+r}} |\{s_i^{z_{2+r}} : z_{2+r} = 1, \ldots, Z_{2+r}\} \cap V_{2+r}'|,$$

which is also modular as a function of V'. Let Ω denote the set of all possible paths in the state space that are chosen by the adversary with nonzero probability such that the termination of the path is at some destination in $\mathcal{D} = \cup_{j=1}^{M} \mathcal{D}_j$. The last term can be written as

$$\sum_{\omega \in \Omega} \pi(\omega) \sum_{j=1}^{M} (p_T(j;\omega)\alpha^D + p_R(j;\omega)\beta_j^D),$$

where $p_T(j;\omega)$ (resp. $p_R(j;\omega)$) denotes the probability that the adversarial flow is detected by the defender at the j^{th} stage (resp. reaches some destination in stage j) when the sample path is ω and the defender strategy is $\mathbf{p}_D(V')$ (the V' is omitted from the notation for simplicity). $\pi(\omega)$ denotes the probability of selecting the path ω. Let $g(\omega; V')$ denote the probability with which adversary

evades detection along path ω when the defender's strategy is $\mathbf{p}_D(V')$. Since the last destination that is reached before dropping out is determined by the choice of path (denote this destination $j(\omega)$), we have

$$\sum_{j=1}^{M}(p_T(j;\omega)\alpha^D+p_R(j;\omega)\beta_j^D)=g(\omega;V')\alpha^D+(1-g(\omega;V'))\beta_{j(\omega)}^D$$
$$=g(\omega;V')(\alpha^D-\beta_{j(\omega)}^D)+\beta_{j(\omega)}^D.$$

Since $\alpha^D-\beta_{j(\omega)}^D\geq 0$ and $\beta_{j(\omega)}^D$ is independent of $\mathbf{p}_D(V')$, it suffices to show that $g(\omega;V')$ is submodular as a function of V'. Let $V'\subseteq V''$ with $V_r'\subseteq V_r''$ and $s_i^{z_r}\notin V_r''$, for any $r\in\{1,\ldots,2+N\}$. We can write $g(\omega;V'')$

$$\begin{aligned}
&= 1-\left[\prod_{\substack{s_{i_k}\in\omega:\\ i_k=i}}(1-\prod_{r=1}^{2+N}\mathbf{p}_D^r(s_{i_k}))\right]\left[\prod_{\substack{s_{i_k}\in\omega:\\ i_k\neq i}}(1-\prod_{r=1}^{2+N}\mathbf{p}_D^r(s_{i_k}))\right]\\
&= 1-\delta(V'')(1-\prod_{r=1}^{2+N}\mathbf{p}_D^r(s_i))^{c(s_i;\omega)},
\end{aligned}$$

where $\mathbf{p}_D^r(s_{i_k})$, for $r=1,\ldots,2+N$, denotes the probabilities of selecting node s_{i_k} as tag source, as tag sink, and selecting security rules under the policy $\mathbf{p}_D(V')$, respectively, and

$$\delta(V'')=\prod_{\substack{s_{i_k}\in\omega:\\ i_k\neq i}}(1-\prod_{r=1}^{2+N}\mathbf{p}_D^r(s_{i_k})),\quad c(s_i;\omega)=|\{s_{i_k}\in\omega:i_k=i\}|.$$

Hence, $g(\omega;\left\{V_r''\cup\{s_i^{z_r}\}\right\}_{r=1}^{2+N})-g(\omega;V'')$

$$\begin{aligned}
&= 1-\delta(V'')(1-\textstyle\prod_{r=1}^{2+N}(\mathbf{p}_D^r(s_i;V'')+\frac{1}{Z_r}))^{c(s_i;\omega)}-\\
&\quad (1-\delta(V'')(1-\prod_{r=1}^{2+N}\mathbf{p}_D^r(s_i;V''))^{c(s_i;\omega)})\\
&= \delta(V'')\Big[(1-\prod_{r=1}^{2+N}\mathbf{p}_D^r(s_i;V''))^{c(s_i;\omega)}-\\
&\quad (1-\prod_{r=1}^{2+N}(\mathbf{p}_D^r(s_i;V'')-\frac{1}{Z_r}))^{c(s_i;\omega)}\Big]
\end{aligned}$$

From Eq. (4.9) and the definition of $\mathbf{P}_{D}(V')$, when $V'_r \subseteq V''_r$, $\mathbf{p}^r_D(s_i; V') \leq \mathbf{p}^r_D(s_i; V'')$, and hence

$$(1 - \prod_{r=1}^{2+N} \mathbf{p}^r_D(s_i; V''))^{c(s_i;\omega)} - (1 - \prod_{r=1}^{2+N} (\mathbf{p}^r_D(s_i; V'') - \frac{1}{Z_r}))^{c(s_i;\omega)}$$
$$\leq (1 - \prod_{r=1}^{2+N} \mathbf{p}^r_D(s_i; V'))^{c(s_i;\omega)} - (1 - \prod_{r=1}^{2+N} (\mathbf{p}^r_D(s_i; V') - \frac{1}{Z_r}))^{c(s_i;\omega)}.$$

Furthermore, $V'_r \subseteq V''_r$ for $r = \{1, \ldots, 2+N\}$ implies $\delta(V') \geq \delta(V'')$. Hence

$$g(\omega; \left\{V'_r \cup \{s_i^{z_r}\}\right\}_{r=1}^{2+N}) - g(\omega; V')$$
$$\geqslant g(\omega; \left\{V''_r \cup \{s_i^{z_r}\}\right\}_{r=1}^{2+N}) - g(\omega; V''),$$

completing the proof of submodularity. □

Submodularity of $f(V')$ implies the following.

Proposition 4.3.8. There exists an algorithm that is guaranteed to select a set $V^\star$ satisfying $f(V^\star) \geq \frac{1}{2} \max \{f(V') : V' \subseteq V\}$ within $O(NZ)$ evaluations of U^D, where $Z = \sum_{r=1}^{2+N} Z_r$.

Proof. The proof follows from submodularity of V' and [32]. □

4.3.2 Solution to DIFT-APT Game for Detecting Single-Stage Cyber Threats

In this section, we focus on the case where there is only a single attack stage and provide a solution for the DIFT game. Recall that in a single-stage attack, the attacker's objective is to choose transitions in the IFG to reach a target node. Our approach to solving the game is based on a minimum capacity cut-set formulation on a flow network constructed for the IFG followed by solving a bimatrix game. Here, $M = 1$ and hence we drop the notation for the stage in this section.

Min-Cut Formulation

For a flow-network $\mathcal{F}$, a *cut* is defined below.

Definition 4.3.9. In a flow-network $\mathcal{F}$ with vertex and directed edge sets $V_{\scriptscriptstyle F}$ and $E_{\scriptscriptstyle F}$ respectively, for a subset $\hat{\mathcal{S}} \subset V_{\scriptscriptstyle F}$ the cut induced by $\hat{\mathcal{S}}$ is a subset of edges $\kappa(\hat{\mathcal{S}}) \subset E_{\scriptscriptstyle F}$ such that for every $(u, v) \in \kappa(\hat{\mathcal{S}})$, $|\{u, v\} \cap \hat{\mathcal{S}}| = 1$.

The set $\kappa(\hat{\mathcal{S}})$ consists of all edges whose one end point is in $\hat{\mathcal{S}}$. Given a flow-network $\mathcal{F} = (V_{\mathcal{F}}, E_{\mathcal{F}})$ with source-sink pair $(s_{\mathcal{F}}, t_{\mathcal{F}})$ and edge capacity vector $c_{\mathcal{F}} : E_{\mathcal{F}} \to \mathbb{R}_+$, the cost of a cut $\kappa(\hat{\mathcal{S}})$, is defined as the sum of the costs of the edges in the cut

$$c_{\mathcal{F}}(\kappa(\hat{\mathcal{S}})) = \sum_{e \in \kappa(\hat{\mathcal{S}})} c_{\mathcal{F}}(e). \tag{4.10}$$

The objective of the *(source-sink)-min-cut problem* is to find a cut $\kappa(\hat{\mathcal{S}}^\star)$ of $\hat{\mathcal{S}}^\star$ such that $c_{\mathcal{F}}(\kappa(\hat{\mathcal{S}}^\star)) \leqslant c_{\mathcal{F}}(\kappa(\hat{\mathcal{S}}))$ for any cut $\kappa(\hat{\mathcal{S}})$ of $\hat{\mathcal{S}}$ satisfying $s_{\mathcal{F}} \in \hat{\mathcal{S}}$ and $t_{\mathcal{F}} \notin \hat{\mathcal{S}}$. The (source-sink)-min-cut problem is well studied and there exist different algorithms that find the maximum flow $f^\star$ in polynomial time (polynomial in $|V_{\mathcal{F}}|$ and $|E_{\mathcal{F}}|$) [100]. Given an information flow graph $\mathcal{G}$, we first construct the flow-network $\mathcal{F}$.

Pseudocode describing the construction of $\mathcal{F} = (V_{\mathcal{F}}, E_{\mathcal{F}})$ is given in Algorithm 3. The vertex set of $\mathcal{F}$ consists of two nodes s_i and s_i' corresponding to each node s_i in the information flow graph $\mathcal{G}$ and additional vertices $s_{\mathcal{F}}, t_{\mathcal{F}}$ (Step 2). Thus $|V_{\mathcal{F}}| = 2N + 2$. The directed edge set of $\mathcal{F}$ consists of all edges in the information flow graph ($\bar{E}_{\mathcal{G}}$), edges corresponding to the nodes in the information flow graph ($\hat{E}_{\mathcal{G}}$), edges connecting source node $s_{\mathcal{F}}$ to all nodes in the set λ (E_λ), and edges connecting all destination nodes to the sink node $t_{\mathcal{F}}$ ($E_{\mathcal{D}}$) (Step 3). The capacity vector $c_{\mathcal{F}}$ is defined in such a way that all edges except the edges corresponding to nodes in $\mathcal{G}$ have infinite capacity. The capacities of the edges in $\hat{E}_{\mathcal{G}}$ are defined as the sum of cost for selecting those nodes as tag source and tag sink since the costs of selecting the security rules do not depend on the node (Step 4). Hence, a minimum capacity edge in $\mathcal{F}$ corresponds to a node in $\mathcal{G}$ that has the minimum cost of tagging and trapping. Let $\kappa(\hat{\mathcal{S}}^\star)$ denote an optimal solution to the (source-sink)-min-cut problem on $\mathcal{F}$. Since $\hat{E}_{\mathcal{G}}$ is a cut and $\sum_{e \in \hat{E}_{\mathcal{G}}} c_{\mathcal{F}}(e) < \infty$, $\kappa(\hat{\mathcal{S}}^\star) \subset \hat{E}_{\mathcal{G}}$ Then, the min-cut nodes is given by

$$\hat{\mathcal{S}}^\star := \{s_i : (s_i, s_i') \in \kappa(\hat{\mathcal{S}}^\star)\}. \tag{4.11}$$

The objective of the defender is to optimally select a defense policy such that no adversarial flow reaches from s_0 to some node in $\mathcal{D}$. Assumption 4.2.2 ensures that there is no path where the cost of tagging exceeds the damage of the attack, so that the defender always has an incentive to

Algorithm 3 Pseudocode for constructing the flow-network $\mathcal{F}$ and defender payoff function $U_{D}(\cdot)$

Input: Information flow graph $\mathcal{G}$, costs $\mathcal{C}_{D}, \mathcal{W}_{D}, \gamma_1, \ldots, \gamma_N$

Output: Flow-network $\mathcal{F}$, source, sink nodes: s_{F}, t_{F}, capacity vector c_{F}

1: Construct flow-network $\mathcal{F}$ with vertex set V_{F} and edge set E_{F} as follows:

2: $V_{F} \leftarrow V_{\mathcal{G}} \cup V'_{\mathcal{G}} \cup \{s_{F}, t_{F}\}$, where $V_{\mathcal{G}} = \{s_1, \ldots, s_N\}$, $V'_{\mathcal{G}} = \{s'_1, \ldots, s'_N\}$, and $s_{F} = s_0$

3: $E_{F} \leftarrow \bar{E}_{\mathcal{G}} \cup \hat{E}_{\mathcal{G}} \cup E_{\lambda} \cup E_{D}$, where $\bar{E}_{\mathcal{G}} = \{(s'_i, s_j) : (s_i, s_j) \in E_{\mathcal{G}}\}$, $\hat{E}_{\mathcal{G}} = \{(s_i, s'_i) : i = 1, \ldots, N\}$, $E_{\lambda} = \{(s_{F}, s_i) : s_i \in \lambda$, and $E_{D} = \{(s'_i, t_{F}) : s_i \in \mathcal{D}\}$

4: $c_{F}(e) \leftarrow \begin{cases} \infty, & e \in \bar{E}_{\mathcal{G}} \cup E_{\lambda} \cup E_{D} \\ \mathcal{C}_{D}(s_i) + \mathcal{W}_{D}(s_i), & e \in \hat{E}_{\mathcal{G}} \end{cases}$

perform security analysis in at least one location along each path. In other words, the defender ensures that no flow from s_{F} reaches t_{F} without getting detected. For achieving this the defender's policy must have a nonzero probability of tag and trap for at least one node in all possible paths from s_{F} to t_{F}. For any adversary policy, the best possible choice for the defender is to tag and trap at a node that has the minimum total cost $\mathcal{C}_{D}(\cdot) + \mathcal{W}_{D}(\cdot)$. An attack path is a directed path from s_0 to some node in $\mathcal{D}$ formed by a sequence of transitions of the adversary. The probability of an attack path under an adversary strategy is the product of the probabilities of all transitions along that path. The adversary plans its transitions to obtain an attack path with least probability of detection. The result below characterizes Nash equilibria of the single-stage.

Theorem 4.3.10. Let $\hat{\mathcal{S}}^{\star}$ be a min-cut of the flow-network $\mathcal{F} = (V_{F}, E_{F})$ constructed in Algorithm 3. Then, at Nash equilibrium for the single-stage attack case, the defender's policy is to tag and trap with equal probability all the nodes in $\hat{\mathcal{S}}^{\star}$. Further, the adversary's strategy is such that each attack path passes through exactly one node in $\hat{\mathcal{S}}^{\star}$.

Before giving the proof of Theorem 4.3.10, we present the following lemma that establishes the first main argument in proving the theorem.

Lemma 4.3.11. Let $\Omega_{\mathcal{D}}$ be the set of all paths in $\mathcal{G}$ from s_0 to some node in $\mathcal{D}$ under any adversary

policy. Then, for a defender policy that assign tag and trap at all nodes in the min-cut $\hat{\mathcal{S}}^\star$ and does not tag and trap any other node, the best response of the adversary is a sequence of transitions such that any attack path (or set of paths if using a mixed policy) passes through exactly one node which is tag source and trap.

Proof. Consider a policy of the defender where all the nodes in the min-cut, i.e., $\hat{\mathcal{S}}^\star$, are tagged and assigned as traps. Note that, all paths in $\Omega_{\mathcal{D}}$ pass through some node in $\hat{\mathcal{S}}^\star$. We prove the argument through a contradiction. Suppose that there exists a path $\omega \in \Omega_{\mathcal{D}}$ such that there are two nodes, say s_i, s_r, in path ω with nonzero probability of tag and trap. Without loss of generality, assume that in ω there exists a directed path from s_i to s_r. Now we show that $\pi(\omega) = 0$, where $\pi(\omega)$ is the probability with which the adversary chooses path ω. Note that $s_i, s_r \in \hat{\mathcal{S}}^\star$. Since $\hat{\mathcal{S}}^\star$ corresponds to a min-cut, there exist paths in $\Omega_{\mathcal{D}}$ that have node s_i in it but not s_r, and vice-versa. Hence for an adversary whose current state is s_i, there exists a path from s_i to some node in $\mathcal{D}$ that guarantees the win of the adversary. The transition probability from s_i to a node in ω that will lead to some node in $\mathcal{D}$ through s_r is zero as this path has lower adversary payoff. This gives $\pi(\omega) = 0$ and completes the proof. □

The following result proves that for any adversary policy the best response of the defender is to tag and trap at one node in every attack path under that adversary policy.

Lemma 4.3.12. Let $\Omega_{\mathcal{D}}$ denote the set of all paths from s_0 to some node in $\mathcal{D}$ under any adversary policy. If the defender's policy be such that the probability of detecting the adversary is the same for all $\omega \in \Omega_{\mathcal{D}}$, then the best response of the defender is always to tag at exactly one node in every $\omega \in \Omega_{\mathcal{D}}$.

Proof. Given $(1 - p(\omega))$'s are same for all $\omega \in \Omega_{\mathcal{D}}$. Consider a defender's policy $\mathbf{p}_D$ in which exactly one node in every $\omega \in \Omega_{\mathcal{D}}$ is chosen as the tag source and tag sink. Assume that the defender policy is modified to $\mathbf{p}'_D$ such that more than one node in some path has nonzero tag and trap probability. Note that, such a $\mathbf{p}'_D$ exists as all paths $\omega \in \Omega_{\mathcal{D}}$ at least have two states in it where the defender can perform security analysis, since destinations are assumed to be at least one node

away from the entry points. This variation updates the probabilities of nodes in a set of paths in $\Omega_{\mathcal{D}}$. For $\mathbf{p}'_{D}$ to be a best response, $U_{D}(\mathbf{p}'_{D}, \mathbf{p}_{A}) \geqslant U_{D}(\mathbf{p}_{D}, \mathbf{p}_{A})$. The defender's payoff is given by

$$U_{D}(\mathbf{p}_{D}, \mathbf{p}_{A}) = \sum_{\omega \in \Omega_{\mathcal{D}}} \pi(\omega) \Big[(1 - p(\omega))\alpha^{D} + p(\omega)\beta^{D} + \sum_{s_i \in \omega} [\mathbf{p}^{1}_{D}(s_i)\mathcal{C}_{D}(s_i) + \mathbf{p}^{2}_{D}(s_i)\mathcal{W}_{D}(s_i) + \sum_{r=1}^{N} \mathbf{p}^{2+r}_{D}(s_i)\gamma_r] \Big]$$

The terms in U_{D} that correspond to α^{D} and β^{D} are the same in both cases as $p(\omega)$'s are equal for all possible paths. Hence the terms in U_{D} differ in the terms corresponding to $\mathcal{C}_{D}$, $\mathcal{W}_{D}$, and γ_r's. Note that defender's probabilities (policy) at two nodes in a path are dependent due to the constraint on $p(\omega)$. Hence for every path whose probabilities are modified, $\sum_{s_i \in \omega} [\mathbf{p}'^{1}_{D}(s_i)\mathcal{C}_{D}(s_i) + \mathbf{p}'^{2}_{D}(s_i)\mathcal{W}_{D}(s_i) + \sum_{r=1}^{N} \mathbf{p}'^{r}_{D}(s_i)\gamma_r] < \sum_{s_i \in \omega} [\mathbf{p}^{1}_{D}(s_i)\mathcal{C}_{D}(s_i) + \mathbf{p}^{2}_{D}(s_i)\mathcal{W}_{D}(s_i) + \sum_{r=1}^{N} \mathbf{p}^{r}_{D}(s_i)\gamma_r]$, as the probability in the single node case is less than the sum of the probabilities of the more than one node case as the events are dependent and the $\mathcal{C}_{D}$ and $\mathcal{W}_{D}$ values are also the least possible (γ_r's are equal at all nodes in the information flow graph). This implies $U_{D}(\mathbf{p}'_{D}, \mathbf{p}_{A}) < U_{D}(\mathbf{p}_{D}, \mathbf{p}_{A})$. Therefore, there exists no best response for the defender which has more than one node with nonzero tag and trap probability in a path, if $p(\omega)$'s are equal for all $\omega \in \Omega_{\mathcal{D}}$. □

The result below deduces a property of the best response of the adversary which along with Lemma 4.3.12 establishes the final main argument to prove Theorem 4.3.10.

Lemma 4.3.13. Let $\Omega_{\mathcal{D}}$ denote the set of all paths from s_0 to some node in $\mathcal{D}$ under any adversarial policy. Let the defender's policy be such that the probability of detecting the adversary is the same for all $\omega \in \Omega_{\mathcal{D}}$. Then the best response of the defender is to tag and trap the flows at the min-cut of the flow-network $\mathcal{F}$ constructed in Algorithm 3.

Proof. By Lemma 4.3.12 the best response of the defender is to tag and trap at one node in every attack path. Note that all attack paths chosen under $\mathbf{p}_{A}$ pass through some node in the min-cut. Assigning nonzero probability of tag and trap at the nodes in $\hat{\mathcal{S}}^{\star}$, all possible attack paths have some nonzero probability of getting detected. We prove the result using a contradiction argument. Suppose that it is not the best response of the defender to tag and trap the nodes in $\hat{\mathcal{S}}^{\star}$. Then, there

exists a subset of nodes $\hat{\mathcal{S}} \subset \mathcal{S}$ such that $\sum_{s_i \in \hat{\mathcal{S}}} \mathcal{C}_{D}(s_i) + \mathcal{W}_{D}(s_i) < \sum_{s_r \in \hat{\mathcal{S}}^\star} \mathcal{C}_{D}(s_r) + \mathcal{W}_{D}(s_r)$ and all possible paths from s_0 to nodes in $\mathcal{D}$ pass through some node in $\hat{\mathcal{S}}$. Then, $\hat{\mathcal{S}}$ is a (source-sink)-cut-set and let $\kappa(\hat{\mathcal{S}}) := \{(s_i, s_i') : s_i \in \hat{\mathcal{S}}\}$. Then, $\kappa(\hat{\mathcal{S}})$ is a cut set and $c_{F}(\kappa(\hat{\mathcal{S}})) < c_{F}(\kappa(\hat{\mathcal{S}}^\star))$. This contradicts the fact that $\kappa(\hat{\mathcal{S}}^\star)$ is an optimal solution to the (source-sink)-min-cut problem. Hence the best response of the defender is to tag and trap only the nodes in $\hat{\mathcal{S}}^\star$. □

Now we present the proof of Theorem 4.3.10.

Proof of Theorem 4.3.10: Lemma 4.3.11 proves that the best response of the adversary is any sequence of transitions that gives a path (or set of paths if mixed policy) that passes through exactly one node that is a tag source and a trap if the defender's policy is to tag at the min-cut nodes. Lemma 4.3.13 concludes that the best response of the defender is to tag and trap the adversary at the nodes in the min-cut of $\mathcal{F}$, provided the probability of detecting the adversary in all $\omega \in \Omega_{\mathcal{D}}$ are equal. This implies that, if the detection probabilities ($(1 - p(\omega))$'s) are equal at NE, then the defender's policy at NE will tag and trap the nodes in the min-cut and the adversary will choose an attack path such that it passes through exactly one node that is tagged and also a trap. Now we show that the detection probabilities are indeed equal at NE.

Consider any unilateral deviation in the policy of the adversary. Let $\pi(\omega)$'s for $\omega \in \Omega$ be modified due to change in transition probabilities of the adversary such that the updated probabilities of the attack paths are $\pi(\omega_i) + \epsilon_i$, for $i = 1, \ldots, |\Omega|$. Here, ϵ_i's can take positive values, negative values or zero such that $\sum_{i=1}^{|\Omega|} \epsilon_i = 0$. Consider two arbitrary paths, say ω_1 and ω_2, such that a unilateral change in the adversary policy changes $\pi(\omega_1)$ and $\pi(\omega_2)$ and the probabilities of the other paths remain unchanged. Without loss of generality, assume that $\pi(\omega_1)$ increases by ϵ while $\pi(\omega_2)$ decreases by ϵ and all other $\pi(\omega)$'s remain the same. As $(\mathbf{p}_{D}, \mathbf{p}_{A})$ is a Nash equilibrium $(\pi(\omega_1) + \epsilon)\Big(p(\omega_1)(\beta^{A} - \alpha^{A}) + \alpha^{A}\Big) + (\pi(\omega_2) - \epsilon)\Big(p(\omega_2)(\beta^{A} - \alpha^{A}) + \alpha^{A}\Big) \leqslant \pi(\omega_1)\Big(p(\omega_1)(\beta^{A} - \alpha^{A}) + \alpha^{A}\Big) + \pi(\omega_2)\Big(p(\omega_2)(\beta^{A} - \alpha^{A}) + \alpha^{A}\Big)$. This implies $(p(\omega_1) - p(\omega_2))(\beta^{A} - \alpha^{A}) \leqslant 0$. As $(\beta^{A} - \alpha^{A}) \geqslant 0$, this implies $(p(\omega_1) - p(\omega_2)) \leqslant 0$. By exchanging the roles of ω_1 and ω_2 and using the same argument one can also show that $(p(\omega_1) - p(\omega_2)) \geqslant 0$. This implies $(p(\omega_1) - p(\omega_2)) = 0$.

Since ω_1 and ω_2 are arbitrary, one can show that for the general case

$$\sum_{i=1}^{|\Omega|} \epsilon_i p(\omega_i) = 0. \tag{4.12}$$

Eq. (4.12) should hold for all possible values of ϵ_i's satisfying $\sum_{i=1}^{|\Omega|} \epsilon_i = 0$. This gives $p(\omega_i) = p(\omega_j)$ for all $i, j \in \{1, \ldots, |\Omega|\}$ at Nash equilibrium. This completes the proof. □

The NE of the game is characterized by a solution of the min-cut problem and the set of transitions of the adversary such that all attack paths have exactly one node which is tagged and also a trap. Moreover, the tag and trap probability of these nodes are equal. Note that, the solution to the min-cut problem is not unique in a general flow graph and hence the NE of the game may not be unique. Results in this subsection conclude that at any NE the defender's policy will tag and trap the nodes in the min-cut with equal detection probability and the adversary chooses its transitions such that in every attack path exactly one node is a tag source and a tag sink.

Matrix Game Analysis for Nash Equilibrium

Table 4.1 Matrix game for single-stage case with disjoint attack paths

Defender \ Adversary	$\hat{s}_1$	$\hat{s}_2$	$\ldots$	$\hat{s}_a$
Not detected	(β^D, β^A)	(β^D, β^A)	$\ldots$	(β^D, β^A)
Detected	$(\alpha^D + \text{cost}(\hat{s}_1), \alpha^A)$	$(\alpha^D + \text{cost}(\hat{s}_2), \alpha^A)$	$\ldots$	$(\alpha^D + \text{cost}(\hat{s}_a), \alpha^A)$

In this subsection, we discuss the matrix-game formulation of the single-stage case give in Table 4.1. We first solve the (source-sink)-min-cut problem on $\mathcal{F}$. Let an optimal solution be $\kappa(\hat{\mathcal{S}}^\star)$. Let the vertex set corresponding to $\kappa(\hat{\mathcal{S}}^\star)$ be $\hat{\mathcal{S}}^\star = \{\hat{s}_1, \ldots, \hat{s}_a\}$, where $\hat{\mathcal{S}}^\star := \{s_i : (s_i, s_i') \in \kappa(\hat{\mathcal{S}}^\star)\}$. By Theorem 4.3.10, at NE the defender only tag and trap the nodes in $\hat{\mathcal{S}}^\star$ and adversary chooses transitions such that it passes through only one tag and trap. The attack paths

chosen by the adversary are therefore characterized by $\{\hat{s}_1, \ldots, \hat{s}_a\}$. We denote the probability of selecting an attack path corresponding to the node $\hat{s}_i$ as $\pi(\hat{s}_i)$. Further, let cost$(\hat{s}_i)$ denote the total cost of selecting node $\hat{s}_i$ for conducting security analysis.

Remark 4.3.14. At NE, since any attack path passes through exactly one node in $\hat{\mathcal{S}}^\star = \{\hat{s}_1, \ldots, \hat{s}_a\}$ which indeed has equal probability of tag and trap, without loss of generality, one can consider the action space of the adversary as the set of disjoint paths through $\hat{\mathcal{S}}^\star$ and the adversary strategizes over this set of disjoint paths. Thus in the bimatrix formulation, the strategy of the adversary is to select a path which is uniquely defined by a node in $\hat{\mathcal{S}}^\star$.

Now, we present a result that characterizes the set of NE of the single-stage attack case of the game given in Section 4.2.

Theorem 4.3.15. Consider the dynamic game between $\mathcal{P}_D$ and $\mathcal{P}_A$ where the attack consists of a single stage. Let $\hat{\mathcal{S}}^\star = \{\hat{s}_1, \ldots, \hat{s}_a\}$ be a min-cut node set of the flow network $\mathcal{F}$. Then, solution to the bimatrix-game, given in Table 4.1, gives a Nash equilibrium for the single-stage flow tracking game.

Proof. The defender's payoffs for the two cases in Table 4.1 is

$$U_D(\text{Not detected}) = \sum_{i=1}^{a} \pi(\hat{s}_i)\Big(\beta^D\Big) \tag{4.13}$$

$$U_D(\text{Detected}) = \sum_{i=1}^{a} \pi(\hat{s}_i)\Big(\alpha^D + \text{cost}(\hat{s}_i)\Big) \tag{4.14}$$

The defender randomly chooses to detect the adversary or not if Eqs. (4.13) and (4.14) are equal. This gives

$$\sum_{i=1}^{a} \pi(\hat{s}_i)\Big(\beta^D - \alpha^D - \text{cost}(\hat{s}_i)\Big) = 0. \tag{4.15}$$

There are many possible values of $\pi(\hat{s}_i)$'s for $i = 1, \ldots, a$, that satisfy Eq. (4.15). Each of those solution will give a probability mixture, i.e., $\pi(\hat{s}_i)$'s, for the adversary at a Nash equilibrium.

In order to obtain the probability mixture of the defender, we consider the following in Table 4.1. For every $\hat{s}_i \in \hat{\mathcal{S}}^\star$, one can find the set of nodes that belong to the set λ that has a directed

path to the node $\hat{s}_i$ using a depth first search (DFS) algorithm [43]. Let this set be denoted by $\lambda(\hat{s}_i)$. Then, cost($\hat{s}_i$) $= \mathcal{C}_D(\hat{s}_i) + \mathcal{W}_D(\hat{s}_i) + \sum_{r\in\lambda(\hat{s}_i)} \gamma_r$ for all $\hat{s}_i \in \hat{\mathcal{S}}^\star$. Then the probability of not detecting the adversary in an attack path with min-cut node $\hat{s}_i$ in it is given by $(1 - \mathbf{p}_D^1(\hat{s}_i)\mathbf{p}_D^2(\hat{s}_i)\prod_{r\in\lambda(\hat{s}_i)} \mathbf{p}_D^{2+r}(\hat{s}_i))$.

$$U_A(\hat{s}_i) = \Big(1 - \mathbf{p}_D^1(\hat{s}_i)\,\mathbf{p}_D^2(\hat{s}_i) \prod_{r\in\lambda(\hat{s}_i)} \mathbf{p}_D^{2+r}(\hat{s}_i)\Big)\beta^A + \mathbf{p}_D^1(\hat{s}_i)\,\mathbf{p}_D^2(\hat{s}_i) \prod_{r\in\lambda(\hat{s}_i)} \mathbf{p}_D^{2+r}(\hat{s}_i)\alpha^A, \text{ for } i = 1, \ldots, a. \tag{4.16}$$

The adversary will randomly choose between attack paths that correspond to nodes $\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_a$ only when $U_A(\hat{s}_1) = U_A(\hat{s}_2) = \ldots = U_A(\hat{s}_a)$. Theorem 4.3.10 says that

$$p(\omega) = (1 - \mathbf{p}_D^1(\hat{s}_1)\mathbf{p}_D^2(\hat{s}_1) \prod_{r\in\lambda(\hat{s}_1)} \mathbf{p}_D^{2+r}(\hat{s}_1)) = \ldots = (1 - \mathbf{p}_D^1(\hat{s}_a)\mathbf{p}_D^2(\hat{s}_a) \prod_{r\in\lambda(\hat{s}_a)} \mathbf{p}_D^{2+r}(\hat{s}_a)).$$

This implies $U_A(\hat{s}_1) = U_A(\hat{s}_2) = \ldots = U_A(\hat{s}_a)$.

The defender's payoff is given by

$$U_D(\mathbf{p}_D, \mathbf{p}_A) = \sum_{\omega\in\Omega} \pi(\omega)\Big[(1 - p(\omega))\alpha^D + p(\omega)\beta^D + \sum_{s_i\in\omega}[\mathbf{p}_D^1(s_i)\mathcal{C}_D(s_i) + \mathbf{p}_D^2(s_i)\mathcal{W}_D(s_i) + \sum_{r=1}^{N} \mathbf{p}_D^{2+r}(s_i)\gamma_r]\Big]$$

At Nash equilibrium with respect to the solution $\hat{\mathcal{S}}^\star$ of the (source-sink)-min-cut game, changing $\mathbf{p}_D^{2+r}(\hat{s}_i)$ for any value of $r \in \{1, \ldots, N\}$ and for one node, say $\hat{s}_i \in \hat{\mathcal{S}}^\star$, will not improve the payoff U_D. Firstly, let us assume that the tagging probability at $\hat{s}_i$ changes from $\mathbf{p}_D^1(\hat{s}_i)$ to $\mathbf{p'}_D^1(\hat{s}_i)$. By equilibrium condition $U_D(\mathbf{p}_D, \mathbf{p}_A) \geqslant U_D(\mathbf{p'}_D, \mathbf{p}_A)$. Note that by Lemma 4.3.11 each node with nonzero defender's probability will lie in exactly one chosen path by the adversary. Hence

$$\pi(\hat{s}_i)\Big[p(\omega)(\beta^D - \alpha^D) + \alpha^D + \mathbf{p}_D^1(\hat{s}_i)\mathcal{C}_D(\hat{s}_i)\Big]$$

$$\geqslant \pi(\hat{s}_i)\Big[p'(\omega)(\beta^D - \alpha^D) + \alpha^D + \mathbf{p'}_D^1(\hat{s}_i)\mathcal{C}_D(\hat{s}_i)\Big], \text{ and}$$

$$\pi(\hat{s}_i)\Big[(p(\omega) - p'(\omega))(\beta^D - \alpha^D) + (\mathbf{p}_D^1(\hat{s}_i) - \mathbf{p'}_D^1(\hat{s}_i))\mathcal{C}_D(\hat{s}_i)\Big] \geqslant 0.$$

Here

$$p(\omega) - p'(\omega) = \underbrace{\Big(\mathbf{p}_D^2(\hat{s}_i) \prod_{r \in \lambda(\hat{s}_i)} \mathbf{p}_D^{2+r}(\hat{s}_i)\Big)}_{\varphi_1(\hat{s}_i)} \Big(\mathbf{p'}_D^1(\hat{s}_i) - \mathbf{p}_D^1(\hat{s}_i)\Big).$$

This gives

$$\Big[\pi(\hat{s}_i)\Big(\varphi_1(\hat{s}_i)(\beta^D - \alpha^D) - \mathcal{C}_D(\hat{s}_i)\Big)\Big](\mathbf{p'}_D^1(\hat{s}_i) - \mathbf{p}_D^1(\hat{s}_i)) \geqslant 0.$$

The term $\pi(\hat{s}_i)\Big[\varphi_1(\hat{s}_i)(\beta^D - \alpha^D) - \mathcal{C}_D(s_i)\Big]$ is independent of the change in the tagging probability at $\hat{s}_i$ and the value is either positive or negative. By the equilibrium assumption, $\pi(\hat{s}_i)\Big[\varphi_1(\hat{s}_i)(\beta^D - \alpha^D) - \mathcal{C}_D(\hat{s}_i)\Big] = 0$ (since $(\mathbf{p'}_D^1(\hat{s}_i) - \mathbf{p}_D^1(\hat{s}_i))$ can be made positive or negative and the inequality must hold for both cases). As $\pi(\hat{s}_i) \neq 0$, this implies

$$\varphi_1(\hat{s}_i) = \frac{\mathcal{C}_D(\hat{s}_i)}{(\beta^D - \alpha^D)}.$$

By varying the tag sink selection probability, i.e., changing $\mathbf{p}_D^2(\hat{s}_i)$ to $\mathbf{p'}_D^2(\hat{s}_i)$, we get

$$\varphi_2(\hat{s}_i) = \frac{\mathcal{W}_D(\hat{s}_i)}{(\beta^D - \alpha^D)}, \text{ where } \varphi_2(\hat{s}_i) := \Big(\mathbf{p}_D^1(\hat{s}_i) \prod_{r \in \lambda(\hat{s}_i)} \mathbf{p}_D^{2+r}(\hat{s}_i)\Big).$$

Similarly, by varying the probability of selection each of the rules at $\hat{s}_i$, for $r \in \{1, \ldots, N\}$,

$$\varphi_{r+2}(\hat{s}_i) = \frac{\gamma_r}{(\beta^D - \alpha^D)}.$$

Taking logarithms of $\varphi_1(\hat{s}_i), \ldots, \varphi_{2+N}(\hat{s}_i)$, for a node $\hat{s}_i \in \mathcal{S}$, we get $2 + N$ independent linear equations with $2 + N$ unknowns. Thus there exists a unique solution to this set of equations which indeed gives the defender's policy at Nash equilibrium. Thus solution to the matrix-game in Table 4.1 gives a NE to the single-stage case. □

This completes the discussion on the NE for the flow tracking game when the attack consists of a single stage.

4.3.3 Solution to DIFT-APT Game for Detecting Multi-Stage Cyber Threats

Solving for Nash equilibrium in non-zero sum, imperfect information game settings is generally known to be computationally difficult. In this section, we present an efficient algorithm to compute a locally optimal correlated equilibrium [17, 102] of the game introduced in Section 4.2.

Formal definitions for the correlated equilibrium and the local correlated equilibrium are given in the Definitions 4.3.3 and 4.3.4, respectively. Intuitively correlated equilibrium can be viewed as a general distribution over a set of strategy profiles such that if an impartial mediator privately recommends actions to each player from this distribution, then no player has an incentive to choose a different strategy. The correlated equilibrium has several advantages [17, 102] : (1) it is guaranteed to always exist and (2) it can be found in polynomial time (i.e., computing a correlated equilibrium only requires solving a linear program whereas solving a Nash equilibrium requires finding its fixed point).

In order to find locally optimal correlated equilibrium solutions, we map our two-player game model into a game with $((M+2)N+\Lambda+1)$ players, where $\Lambda = \sum_{i=1}^{N} \Lambda(s_i)$ with $s_i \in V_{\mathcal{G}}$ and $\Lambda(s_i)$ stands for the total number of security rules associated with the node $s_i \in V_{\mathcal{G}}$ in the information flow graph. Then the adversary's strategy is represented by $(MN + 1)$ players, MN of which represents the adversary's actions at every node $s_i \in V_{\mathcal{G}}$ and for a specified stage $j \in \{1, \ldots, M\}$ and one adversarial player acting on the pseudo node, s_0, whose strategy decides the entry point chosen by the adversary into the system. Similarly, the defender's strategy is represented by $\sum_{i=1}^{N} \Lambda(s_i) + 2N$ players, each one of the $\Lambda(s_i) + 2$ defender player represents the defender's strategy (tag selection, trap selection, and selection of security rules) at a single node s_i.

Formally, we consider a set of players $\{\mathcal{P}^{A_{ij}} : i = 1, \ldots, N, j = 1, \ldots, M\} \cup \{\mathcal{P}^{D_i} : i = 1, \ldots, 2N + \Lambda\} \cup \{\mathcal{P}^{s_0}\}$. Each of the players in $\mathcal{P}^{A_{ij}}$ has action space $\mathcal{A}^{A_{ij}} = \mathcal{N}(s_i)$, each player in $\mathcal{P}^{D_i}$ has action space $\{0, 1\}$ which, depending on the type of defender player, represents whether or not to tag or trap/no trap or selecting or not selecting a specific tag check rule. The player $\mathcal{P}^{s_0}$ has action space λ. We let $\mathbf{a}^D$ denote the set of actions chosen by the players $\{\mathcal{P}^{D_i} : i = 1, \ldots, 2N+\Lambda\}$ and $\mathbf{a}^A$ denote the set of actions chosen by the players $\{\mathcal{P}^{A_{ij}} : i = 1, \ldots, N, j = 1, \ldots, M\} \cup \{\mathcal{P}^{s_0}\}$.

The payoffs of the players from a particular action set are given by

$$\begin{aligned} U^{A_{ij}}(\mathbf{a}_A, \mathbf{a}_D) &= U^A(\mathbf{a}_A, \mathbf{a}_D), \\ U^{D_i}(\mathbf{a}_A, \mathbf{a}_D) &= U^D(\mathbf{a}_A, \mathbf{a}_D), \end{aligned}$$

where U^A and U^D are as defined in Section 4.2. Hence, all adversarial players receive the same payoff U^A, while all defender players receive the payoff U^D. Equivalently, the adversarial players $U^{A_{ij}}$ cooperate in order to maximize the adversary's payoff, while the defender players U^{D_i} attempt to maximize the defender payoff.

Under the solution algorithm, the game is played repeatedly, with each player choosing its action from a probability distribution (mixed strategy) over the set of possible actions. After observing their payoffs, the players update their strategies according to an *internal regret minimization* learning algorithm [35]. A pseudocode of the proposed algorithm for computing correlated equilibrium strategies for both defender and adversary players is given in Algorithm 4.

The algorithm initializes the strategies at each node of the information flow graph to be uniformly random. At each iteration t, an action is chosen for each player according to the probability distribution $\mathbf{p}_{t,n}$ of player n. After observing the actions from other players, the probability distribution $\mathbf{p}_{t,n}$ is updated as follows. For each pair of actions r and s, the new probability distribution $\mathbf{p}_{t,n}^{r\to s}$ is generated, in which all of the probability mass allocated to action r is instead allocated to action s. The expected payoff arising from $\mathbf{p}_{t,n}^{r\to s}$ can be interpreted as the expected benefit from playing action s instead of r at previous iterations of the algorithm.

For each pair (r, s), a weight $\Delta_{(r,s),t,n}$ is computed that consists of the relative benefit of each distribution $\mathbf{p}_{t,n}^{r\to s}$, i.e., pairs (r, s) such that allocating probability mass from r to s produces a larger expected payoff will receive a higher weight. A new distribution $\mathbf{p}_{t,n}$ is then computed based on the weights $\Delta_{(r,s),t,n}$, so that actions that produced a higher payoff for the player will be chosen with increased probability. The algorithm continues until the distributions converge. The convergence of the algorithm is described by the following proposition.

Proposition 4.3.16. Algorithm 4 converges to a local correlated equilibrium of the game introduced in Section 4.2.

Proof. By [35], Algorithm 4 converges to a correlated equilibrium of the $((M + 2)N + \Lambda + 1)$-player game. Equivalently, by Definition 4.3.3, for any s_i and $p'_{D_i} \in [0, 1]$, the joint distribution P

Algorithm 4 Pseudocode of the algorithm for computing correlated equilibrium

Initialize $t \leftarrow 0$
for $n = 1, \ldots, (M+2)N + \Lambda + 1$ **do**
    $\mathbf{p}_{t,n} \leftarrow$ uniform distribution over set of actions
end for
while $||\mathbf{p}_t - \mathbf{p}_{t-1}|| > \epsilon$ **do**
    for $n = 1, \ldots, (M+2)N + \Lambda + 1$ **do**
        $a_{t,n} \leftarrow$ action chosen from distribution $\mathbf{p}_{t,n}$
    end for
    for $n = 1, \ldots, (M+2)N + \Lambda + 1$ **do**
        $a_{t,-n} \leftarrow (a_{t,l} : l \neq n)$
        for all (r, s) actions of player n **do**
            $\mathbf{p}_{t,n}^{r \to s} \leftarrow \mathbf{p}_{t,n}$
            $\mathbf{p}_{t,n}^{r \to s}(r) \leftarrow 0$
            $\mathbf{p}_{t,n}^{r \to s}(s) \leftarrow \mathbf{p}_{t,n}(r) + \mathbf{p}_{t,n}(s)$
            $\Delta_{(r,s),t,n} \leftarrow \frac{\exp\left(\eta \sum_{u=1}^{t-1} \mathbf{E}(U^n(\mathbf{p}_{u,n}^{r \to s}, a_{u,-n}))\right)}{\sum_{(x,y):x \neq y} \exp\left(\eta \sum_{u=1}^{t-1} \mathbf{E}(U^n(\mathbf{p}_{u,n}^{x \to y}, a_{u,-n}))\right)}$
            $\mathbf{p}_{t,n} \leftarrow$ fixed point of equation $\mathbf{p}_{t,n} = \sum_{(i,j):i \neq j} \mathbf{p}_{t,n}^{r \to s} \Delta_{(r,s),t,n}$
        end for
    end for
    $t \leftarrow t + 1$
end while

returned by the algorithm satisfies

$$\mathbf{E}(U^{D_i}(p_{D_i}, \mathbf{p}_{D_{-i}}, \mathbf{p}_A)) \geq \mathbf{E}(U^{D_i}(p'_{D_i}, \mathbf{p}_{D_{-i}}, \mathbf{p}_A)). \tag{4.17}$$

Since the payoff U^{D_i} is equal to U^D for all $i \in \{1, \ldots, N\}$, Eq. (4.17) is equivalent to

$$\mathbf{E}(U^D(p_{D_i}, \mathbf{p}_{D_{-i}}, \mathbf{p}_A)) \geq \mathbf{E}(U^D(p'_{D_i}, \mathbf{p}_{D_{-i}}, \mathbf{p}_A)). \tag{4.18}$$

Similarly, for any $s_i^j \in \{\mathcal{S} \times \{1, \ldots, M\}\} \cup \{s_0^1\}$ and any $p'_{A_{ij}}$, we have

$$\mathbf{E}(U^{A_{ij}}(\mathbf{p}_D, \mathbf{p}_{A_{-ij}}, p_{A_{ij}})) \geq \mathbf{E}(U^{A_{ij}}(\mathbf{p}_D, \mathbf{p}_{A_{-ij}}, p'_{A_{ij}})) \tag{4.19}$$

which is equivalent to

$$\mathbf{E}(U^{A}(\mathbf{p}_D, \mathbf{p}_{A_{-ij}}, p_{A_{ij}})) \geq \mathbf{E}(U^{A}(\mathbf{p}_D, \mathbf{p}_{A_{-ij}}, p'_{A_{ij}})) \tag{4.20}$$

Equations (4.18) and (4.20) imply that the output of Algorithm 4 satisfies the conditions of Definition 4.3.4 and hence is a local correlated equilibrium. □

The following Proposition provides the complexity analysis for the proposed algorithm.

Proposition 4.3.17. With probability $(1 - \zeta)$, Algorithm 4 returns an ϵ-correlated[2] equilibrium using $O\left(\frac{N^2(M+2)+N(\Lambda+1)}{\epsilon^2} \ln\left(\frac{N^2(M+1)+N(\Lambda+1)}{\zeta}\right)\right)$ evaluations of the payoff function.

Proof. By [35, Chapter 7, Section 7.4], learning-based algorithms return an ϵ-correlated equilibrium with probability $(1 - \zeta)$ within $\max_n \frac{16}{\epsilon^2} \ln \frac{N_n K}{\zeta}$ iterations, where N_n is the number of actions for player n and K is the number of players, incurring a total of $\frac{16 N_n K}{\epsilon^2} \ln \frac{N_n K}{\zeta}$ evaluations of the payoff function. In this case, $N_n \leq N$ and $K = (M + 2)N + \Lambda + 1$, resulting in the desired complexity bounds. □

Proposition 8.5.3 shows that convergence of the algorithm is sublinear in the number of nodes, with a total complexity that is quadratic in the number of nodes and linear in the number of stages.

4.4 Simulation Study

In this section, provide experimental validation of our model and simulation results to complement our theoretical results using real-world attack data obtained using Refinable Attack INvestigation system (RAIN) [64], [67] for a three day nation state attack. We implement our model and run Algorithm 4 on the information flow graph generated using the system log data obtained using

[2] An ϵ-correlated equilibrium is a correlated equilibrium at which no player can improve the expected payoff more that ϵ by unilaterally deviating from the strategy.

the RAIN system for day one of the nation state attack. The output of the algorithm gives the defender's policy, i.e., tagging locations, trapping locations and selection of appropriate security rules, at a local equilibrium of the multi-stage game. We also perform sensitivity analysis by varying the cost of defense (i.e., tagging costs, trapping costs and individual security rule selection costs at each node in the underlying information flow graph) for the defender. This analysis enables us to infer the optimal strategies of the players and the sensitivity of the model with respect to cost parameters for a given attack data set (information flow graph with specified destinations for each attack stage).

We present below the details of the attack we consider and the steps involved in the construction of the information flow graph for that attack.

4.4.1 Attack Description

The evaluation was completed on a nation state attack (i.e., state-of-the-art APT attack) orchestrated by a red-team during an adversarial engagement. The engagement was organized by a US government agency (US DARPA). During the engagement, we leveraged RAIN [64] to record the whole-system log. At a high-level the goal of the adversaries' campaign was to steal sensitive proprietary and personal information from the targeted company. The attack is designed to run through three days. We only consider the day 1 log data for our evaluation purposes. Through our extensive analysis, we partitioned the attack in day 1 into four key stages: initial compromise, internal reconnaissance, foothold establishment, and data exfiltration. The initial compromise leveraged a spear-phishing attack, which lead the victim to a website that was hosting ads from a malicious web server. The victim navigated to the website, which exploited a vulnerability in the Firefox browser. Once the attackers compromised the machine, the next stage of the APT leveraged common payoffs to do internal reconnaissance. The goal of this stage was to fingerprint the compromised system to detect running processes and network information. The attackers then established a foothold by writing a malicious program to disk. The malicious program was eventually executed, and established a backdoor, which was used to continuously exfiltrate the companies sensitive data.

The system log data for day 1 of the nation state attack is collected with the annotated entry

points of the attack and the attack destinations corresponding to each stage of the attack. Initial conversion of the attack data into an information flow graph resulted in a coarse-grain graph with $\approx$ 132,000 nodes and $\approx$ 2 million edges. The coarse graph captures the whole system data during the recording time which includes the attack-related data and lots of data related to the system's background processes (noise). Hence coarse graph provides very little security-sensitive (attack-related) information about the underlying system and it is computationally intensive to run our algorithm on such a coarse graph. Without loss of any relevant information, we pruned the coarse graph to extract the security-sensitive information about the system from the data [64]. The resulting information flow graph is called *security-sensitive information sub-graph* [64] and we run our experimental analysis on this graph. The pruning includes the following two major steps:

1. Starting from the provided attack destinations, perform upstream, downstream and point-to-point stream techniques discussed in [64] to prune the coarse information flow graph.

2. Prune the resulting subgraph from Step 1) by combining object nodes (e.g. files, net-flow objects) that belong to the same directories or that use the same network socket.

The resulting pruned information flow graph (see Figure 4.1) consists of 30 nodes ($N = 30$) out of which 8 nodes are identified as attack destination nodes corresponding to each of the 4 stages ($M = 4$) of the day 1 nation state attack. One node related to a net-flow object has been identified as an entry point used for the attack ($|\lambda| = 1$). Note that, even when the sensitive locations in the system are known it may not be feasible to do tagging, trapping, and security analysis at that location (entry point of attack) which is captured in our model by the costs for tagging, trapping, and performing authenticity analysis using a security rule.

4.4.2 Case Study 1: Convergence of the algorithm

In this section, we provide a case study that validates the convergence of the proposed algorithm. For simulation purposes, we assume the fraction of the flows through each node is uniformly distributed. From the system log, one can estimate the distribution of the fraction of flows through

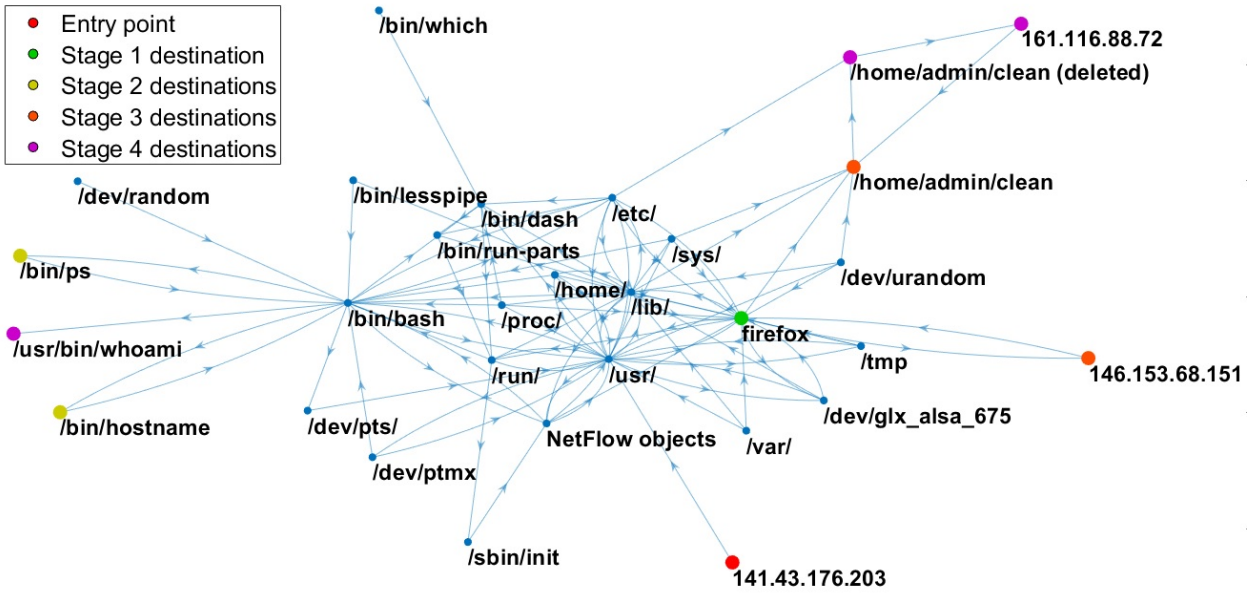
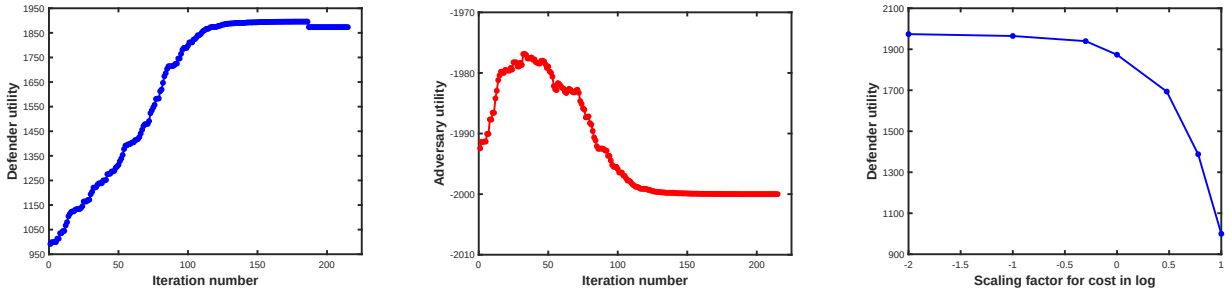


Figure 4.1: Security-sensitive information flow sub-graph for the nation state attack.

each node by averaging the number of events associated with each node with respect to the total number of events occurred during the whole system recording. Figures 4.2(a) and (b) plots the payoff values for both players at each iteration of Algorithm 4 for the above-mentioned parameters. Both defender and adversary payoffs converge within a finite number of iterations.

4.4.3 Case Study 2: Payoff of the defender vs. defense cost

This case study is used to analyze the effect of the cost of defense on the defender's payoff. We use the same game parameters as in Case Study 1. Figure 4.2(c) shows that the expected payoff of the defender starts decreasing exponentially as the cost of defense increases. When the defense cost increases defender incurs more resource cost to maintain the same level of security (maintain low attack success probability by setting up tag sources and traps) in the system or higher costs to keep the defender away from frequently deploying tag sources and traps in the system and hence the attack successes rate increases and the defender's payoff decreases. This result implies that



(a) (b) (c)

Figure 4.2: **(a)** Average payoff of the defender and **(b)** Average payoff of the adversary, at each iteration of Algorithm 4 with cost parameters of the game set as follows: $\beta_1^A = 100, \beta_2^A = 200, \beta_3^A = 500, \beta_4^A = 1200,$ $\alpha^A = -2000, \alpha^D = 2000, \beta_1^D = -100, \beta_2^D = -200, \beta_3^D = -500, \beta_4^D = -1200$. The costs of tagging and trapping at each node are set with a fixed cost value of $c_1 = -50$ and $c_2 = -50$, respectively, multiplied by the fraction of flows through each node in the information graph. The cost of selecting the security rules are set as $\gamma_1 = \ldots = \gamma_N = -50$. For simulation purposes we assume that the fraction of the flows through each node in the information flow graph is uniform. **(c)** Average payoff of defender as a function of the cost for defense. In each realization of the experiment we scale the components of the cost for defense (i.e. costs for tag, trap and tag check rule selection) by a increasing scaler factors 0.01, 0.1, 0.5, 1, 3, 6 and 10. All the other game parameters have been fixed to constant values used in the Case Study 4.4.2.

an optimal selection of the locations for tagging and trapping is critical for a resource-efficient implementation of detection system against APTs.

4.5 Summary

This chapter proposed a game-theoretic framework for cost-effective detection of Advanced Persistent Threats (APTs) that perform a multi-stage attack. We used an information flow tracking-based detection mechanism as APTs continuously introduce information flows in the system. As the defense mechanism is unaware of the stage of the attack and also can not distinguish whether an information flow is malicious or not, the game considered in this chapter has asymmetric

information structure. Further, the resource costs of the attacker and the defender are not the same, resulting in a multi-stage nonzero-sum imperfect information game. We first computed the best responses of both players. For the adversary, we showed that the best response can be obtained using a shortest path algorithm in polynomial time. For the defender, we showed that the payoff function of the defender is submodular for a given adversary strategy and hence a $1/2$-optimal solution to the best response can be found in polynomial time. For solving the game in order to obtain an optimal policy for the defender, we first considered the single-stage attack case and characterized the set of Nash equilibria by proving the equivalence of the dynamic game to a bimatrix-game formulation. We showed that at equilibrium an optimal defense strategy is to choose a min-cut node set corresponding to the information flow graph to conduct the security analysis. Then, we considered the multi-stage case and provided a polynomial time algorithm to find an ϵ-correlated equilibrium, for any given ϵ. We performed experimental analysis of our model on real-data for a three day nation state attack obtained using Refinable Attack INvestigation (RAIN) system.

Chapter 5

DIFT-APT GAMES FOR DETECTING MULTIPLE SINGLE-STAGE CYBER THREATS

5.1 Motivation

The granularity of security analysis for detection and risk assessment varies across different types of attackers [64,97]. For instance, while a fine-grain level analysis to detect an Advanced Persistent Threat (APT) incurs high memory, a coarse-grained analysis with lesser memory suffices to identify cyber criminals [64]. Any modeling framework should accommodate these levels of granularities to address the fundamental trade-off between detection of multiple attackers, which makes the memory allocation problem further challenging.

In this chapter we present an analytical model of a resource constrained DIFT that analyzes a set of information flows to simultaneously detect K attackers in a system. The effectiveness of detection depends on the actions of the attackers and the attackers' chances of achieving their goals depend on the allocation of the defense resources. This strategic interaction motivates a game theoretic modeling as it allows us to investigate the trade-off between detection probability of different attackers and memory allocation of DIFT. The $(K+1)$-player game is dynamic where at each stage the attackers choose their next operation in the system and the defender chooses the type of analysis at each flow which characterizes the allocation of the memory.

We make the following contributions in this chapter:

- We prove that finding an optimal defense strategy against given attackers' strategies is equivalent to maximizing an increasing DR-submodular function subject to a polytope constraint. Using the equivalence, we provide an algorithm to compute an approximate optimal strategy of the defender.

- We show that finding an optimal attacker's strategy, for given strategies of other attackers and the defender, is equivalent to solving a shortest path problem on the information flow graph of the system. Based on this mapping we propose a polynomial-time algorithm for computing an optimal attacker strategy.

- We evaluate the performance of our algorithm on data sets collected using Refinable Attack INvestigation (RAIN) system and analyze the optimal system memory allocation.

5.2 DIFT-APT Game Formulation

In this section, we model the $(K+1)$-multi-player dynamic game, Γ, between the defender player $\mathcal{P}_D$ and the adversarial players $\mathcal{P}_{A_1}, \ldots, \mathcal{P}_{A_K}$. All players act simultaneously at each stage of the game and no player can observe the action of any other player. As a result, the game is an *imperfect* information game. We assume that all players are rational and know the payoff function and the goal of every other player. Hence the game is a *complete* information game. The game evolves in the time $t = 0, 1, 2, \ldots, T$, where $t = T$ denotes the termination time of the game and t is referred to as the stage of the game.

State Space: Let $\bar{\mathcal{S}}$ denote the state space of the game. Then each state $\bar{s} \in \bar{\mathcal{S}}$ is structured as $\bar{s} = (s_1, \ldots, s_W)$. The entries $s_1, \ldots, s_W$ represent the current status of the W tagged information flows in the system as observed by the defender at time t. We let the first K entries in any state $\bar{s}$, i.e., $s_1, \ldots, s_K$, represent the status of the K adversarial flows in the IFG. The defender observes all the entries in $\bar{s}$ but is unaware of whether each tagged flow observed at $s_1, \ldots, s_W$ is benign or adversarial. Each type-k adversary only observes the s_k^{th} entry of a state $\bar{s}$. Notice that the players of the game have asymmetric information on the states of the game. Furthermore, we introduce a new state, $\mathbf{s_0}$, where $s_1 = \ldots = s_W = s_0$, to denote the initial state of the game at time $t = 0$. We augment s_0 into the underlying IFG of the game as a virtual node. The set of outgoing edges from the virtual node, s_0, to each node in the set $\lambda \subset V_{\mathcal{G}}$ (entry points of the K adversaries) models the initial foothold of the K adversaries in the system.

For a state $\bar{s} = (s_1, \ldots, s_W)$, $s_i = v_j$ indicates the arrival of i^{th} information flow to node

$v_j \in V_{\mathcal{G}}$ in the IFG. Here, $i = 1, \ldots, W$ and $j = 1, \ldots, N$. We let $s_i = \phi$, for any $i = 1, \ldots, W$, represent the case where i^{th} information flow observed by the DIFT is no longer being continued in the system (i.e., dropped out). Also, $s_k = \tau$, for any $k = 1, \ldots, K$, represents the case where DIFT has successfully trapped the k^{th} adversarial flow before it reaches its destination node d_k. Moreover, if any $s_i = \phi$ (or $s_k = \tau$) for a state $\bar{s}$ at time t, then that entry s_i (s_k, respectively) will remain at ϕ (τ, respectively) for any future state $\bar{s}'$ at time $t' > t$. Hence, $s_k \in V_{\mathcal{G}} \cup \{\phi, \tau\}$ for all $k = 1, \ldots K$ and $s_i \in V_{\mathcal{G}} \cup \{\phi\}$ for all $i = K+1, \ldots W$. The total number of states in the game is $O((N+2)^K N^{W-K}) + 1$.

A state $\bar{s} = (s_1, \ldots, s_W)$ is a terminal state if $s_k \in \{d_k, \phi, \tau\}$, for all $k \in \{1, \ldots, K\}$. Let the set $\bar{\mathcal{S}}_T \subset \bar{\mathcal{S}}$ denote the set of terminal states in Γ. This implies that at a state $\bar{s} \in \bar{\mathcal{S}}_T$, each of the k adversaries either reached its respective destination d_k in the system or abandoned the attack or is detected by the defender before reaching its goal.

Action Space: The action sets of players are finite. At every state in the game except the terminal states $\bar{s} \in \bar{\mathcal{S}}_T$, adversarial players $\mathcal{P}_{A_1}, \ldots, \mathcal{P}_{A_K}$ decide which out-neighboring node of the IFG to transition to or drop the attack. The defender player decides whether to trap or not to trap each tagged information flow, $s_1, \ldots, s_W$, observed at time t. Moreover, if defender decides to trap a tagged information flow then it must also decide which security policy it should choose to analyze the trapped flow. Let $\mathcal{A}_{A_1}, \ldots, \mathcal{A}_{A_K}$ and $\mathcal{A}_D$ denote the action sets of $\mathcal{P}_{A_1}, \ldots, \mathcal{P}_{A_K}$ and $\mathcal{P}_D$, respectively. Then, at a state $\bar{s} = (s_1, \ldots, s_W)$, $s_k \notin \{d_k, \phi, \tau\}$, $\mathcal{A}_{A_k}(s_k) = \{v_i : (v_j, v_i) \in E_{\mathcal{G}} \text{ and } s_k = v_j\} \cup \varnothing$. Here, the action v_i indicates k^{th} adversary transitioning to a neighboring node in the IFG while $\varnothing$ represents k^{th} adversary dropping out of the game.

Let $\Pi = \{\pi_1, \ldots, \pi_K\}$ denote the set of all K security analysis and $\pi_k \in \Pi$ be the type-k security analysis, where $k = 1, \ldots, K$. Then define $\Pi(\bar{s}) \subseteq \Pi$ to be the set of security analysis feasible in the state $\bar{s}$, i.e., $\pi_k \in \Pi(\bar{s})$ if $s_k \neq \{\tau\}$ in $\bar{s}$. This implies that after successfully detecting a type-k adversarial flow, DIFT can stop analyzing rest of its tagged flows using type-k analysis as the k^{th} attacker is already detected by the defender. Then $\mathcal{A}_D(\bar{s}) = \mathcal{A}_D(s_1) \times \mathcal{A}_D(s_2) \times \ldots \times \mathcal{A}_D(s_W)$, where $\mathcal{A}_D(s_i) = \Pi(\bar{s}) \cup \{0\}$ for $i = 1, \ldots, W$ gives the set of feasible actions for the defender at the current location of the tagged flow, s_i. The action 0 represents the defender not

trapping a tagged flow observed at s_i. At states $\bar{s} \in \bar{\mathcal{S}}_T$, $\mathcal{A}_{A_1}(\bar{s}) = \ldots \mathcal{A}_{A_K}(\bar{s}) = \mathcal{A}_D(\bar{s}) = \emptyset$. In our formulation, the defender is unaware whether an incoming flow is malicious or not and the adversaries do not know whether the adversarial flows are going to get trapped at the nodes they reach at time t. The game captures the information asymmetry between the players as all players take actions simultaneously and no player observe others' actions.

Player Strategies: If we allow all players to use information obtained from previous game stages (i.e., the previous state transitions, previous trapping strategies) to determine their action at each stage, the size of the set of possible strategies for the defender and the adversaries can be very large. To reduce the computational complexity and to model the fact that trapping an adversarial information flow is a lower-level processes with limited computation capability, we restrict our focus to stationary strategies.

Definition 5.2.1. A player strategy is said to be stationary if it depends only on the current state of the player in the game.

We consider mixed strategies in which players randomize over their action spaces. Let $\mathbf{P}_{A_1}, \ldots, \mathbf{P}_{A_k}$ and $\mathbf{P}_D$ denote the set of (mixed) stationary strategies of the players. Strategy of the k^{th} adversary at a state $\bar{s} \in \bar{\mathcal{S}}$ is $p_{A_k}(\bar{s}) \in \mathbf{P}_{A_k} : s_k \rightarrow [0,1]^{|\mathcal{A}_{A_k}(s_k)|}$. The defender's strategy at a state $\bar{s}$ is $p_D(\bar{s}) \in \mathbf{P}_D : \bar{s} \rightarrow [0,1]^{W|\mathcal{A}_{D_k}(\bar{s})|}$. Furthermore, $p_D(\bar{s}) = \begin{bmatrix} p_D^{\bar{s}}(s_1) & \ldots & p_D^{\bar{s}}(s_W) \end{bmatrix}$, where $p_D^{\bar{s}}(s_i) = \begin{bmatrix} p_D^{\bar{s}}(s_i,1) & \ldots & p_D^{\bar{s}}(s_i,K) \end{bmatrix}$ for all $i = 1, \ldots, W$. $p_D^{\bar{s}}(s_i,k)$ denotes the probability of using type-k trapping for a tagged information flow observed at a location corresponding to s_i in the state $\bar{s}$. Here, s_i is the location of the i^{th} tagged information flow observed according to the state $\bar{s}$.

Payoffs: Payoff of an adversarial player consists of three components. An adversarial player $\mathcal{P}_{A_k}$ gets (i) a reward of $\beta_k^A > 0$ for reaching its destination d_k without being detected by the defender, (ii) a penalty of $\alpha_k^A \leqslant 0$ for getting detected by the defender, and (iii) a penalty of $\sigma_k^A \leqslant 0$ for abandoning the attack by dropping out, for $k \in \{1, \ldots, K\}$. Recall that the adversaries are ordered based on their attack capability as $\{\mathcal{P}_{A_1}, \ldots, \mathcal{P}_{A_K}\}$. Thus, $\beta_1^A \leqslant \ldots \leqslant \beta_K^A$, $\alpha_1^A \geqslant \ldots \geqslant \alpha_K^A$, and $\sigma_1^A \geqslant \ldots \geqslant \sigma_K^A$. On the other hand, the defender payoff consists of four components. The defender

player $\mathcal{P}_D$ gets (i) a penalty of $\beta_k^D \leqslant 0$ for k^{th} adversary reaching its destination d_k without being detected by the defender, (ii) a reward of $\alpha_k^D > 0$ for detecting the k^{th} adversary, (iii) a reward of $\sigma_k^D \geqslant 0$ if the k^{th} adversary abandon the attack by dropping out, and (iv) a resource constraint M. Then, $\beta_1^D \geqslant \ldots \geqslant \beta_K^D$, $\alpha_1^D \leqslant \ldots \leqslant \alpha_K^D$, and $\sigma_1^A \leqslant \ldots \leqslant \sigma_K^A$. At every state of the game the resource (memory) constraint should be satisfied. Moreover, the memory requirement to perform security analysis for different types of adversaries are different. The amount of memory required to perform security analysis related to highly capable adversaries is higher than the memory required to deploy security analysis to detect less capable adversaries. Also, the memory required to perform security analysis depends on the node in the IFG as the required memory can vary depending on the type of the process (e.g. process, file or network socket) and the busyness of the process. Let $\mathcal{C}_k(v_i)$ denote the amount of memory required to perform type-k trapping analysis at node $v_i \in V_{\mathcal{G}}$. Then, under a policy $p_D \in \mathbf{P}_D$ at every state $\bar{s} \in \bar{\mathcal{S}}$, $\sum_{i=1}^{W} \sum_{k=1}^{K} p_D^{\bar{s}}(s_i, k)\mathcal{C}_k(s_i) \leqslant M$.

Let $\bar{p}_T(k)$, $\bar{p}_R(k)$, and $\bar{p}_\phi(k)$ denote the probability that adversary $\mathcal{P}_{A_k}$ is detected by the defender $\mathcal{P}_D$, the probability that adversary $\mathcal{P}_{A_k}$ reach destination d_k, and the probability that adversary $\mathcal{P}_{A_k}$ drops out of the game, respectively. Here, $\bar{p}_T(k)$, $\bar{p}_R(k)$, and $\bar{p}_\phi(k)$ are functions of strategies $p_{A_1}, \ldots, p_{A_K}, p_D$. Now we define the payoff functions of the players of Γ. Let $U_{A_1}, \ldots, U_{A_K}, U_D$ denote the payoff functions of the K adversarial players and the defender player, respectively. For a given strategy, p_D and $p_A = \{p_{A_k}\}_{k=1}^K = \{p_{A_1}, \ldots, p_{A_K}\}$, the payoffs $\{U_{A_k}\}_{k=1}^K$, and U_D are

$$U_D(p_A, p_D) = \sum_{k \in \{1,\ldots,K\}} \Big(\bar{p}_T(k)\alpha_k^D + \bar{p}_R(k)\beta_k^D + \bar{p}_\phi(k)\sigma_k^D \Big), \tag{5.1}$$

$$U_{A_k}(p_A, p_D) = \bar{p}_T(k)\alpha_k^A + \bar{p}_R(k)\beta_k^A + \bar{p}_\phi(k)\sigma_k^A. \tag{5.2}$$

Additionally, the defender policy p_D must satisfy the following memory constraint

$$\sum_{i=1}^{W} \sum_{k=1}^{K} p_D^{\bar{s}}(s_i, k)\mathcal{C}_k(s_i) \leqslant M, \text{ for all } \bar{s} \in \bar{\mathcal{S}}. \tag{5.3}$$

In this chapter, our goal is to find best responses of the players which is defined below.

Definition 5.2.2. Let $p_D : \bar{\mathcal{S}} \rightarrow [0,1]^{|\bar{\mathcal{S}}|}$ denote a defender strategy and $p_A : \bar{\mathcal{S}} \rightarrow [0,1]^{|\bar{\mathcal{S}}|}$ denote an adversary strategy. The set of best responses of the defender is given by

$$\text{BR}(p_A) = \arg\max_{p_D} \{U_D(p_D, p_A) : p_D \in [0,1]^{|\mathcal{A}_D|}\}.$$

Similarly, the best responses of the adversary given by

$$\text{BR}(p_D) = \arg\max_{p_A} \{U_A(p_D, p_A) : p_A \in [0,1]^{|\mathcal{A}_A|}\}.$$

In what follows, we focus on finding best responses of the players of the game.

5.3 Best Response Analysis

5.3.1 Best Response of the Defender

For a given strategies $p_{A_1}, \ldots, p_{A_K}$ of K adversarial players, the best response of defender is characterized by the following optimization problem.

Problem 5.3.1. The defender's problem is as follows:
$\max_{p_D \in \mathbf{P}_D} \ \sum_{k\in\{1,\ldots,K\}} \left(\bar{p}_T(k)\alpha_k^D + \bar{p}_R(k)\beta_k^D + \bar{p}_\phi(k)\sigma_k^D\right)$
Subject to: $\sum_{i=1}^{W}\sum_{k=1}^{K} p_D^{\bar{s}}(s_i,k)\mathcal{C}_k(s_i) \leqslant M$, *for all* $\bar{s} \in \bar{\mathcal{S}}$.

There exists a reduction of a general instance of a multi-set cover problem to an instance of Problem 5.3.1 which provides the following hardness result.

Proposition 5.3.2. The defender's best response problem, Problem 5.3.1, is NP-hard.

We omit the proof of Proposition 5.3.2 in the interest of space. Also note that Problem 5.3.1 is a non-convex optimization problem. To this end, we investigate on additional structure of the payoff function of the defender to obtain a tractable solution to Problem 5.3.1.

Definition 5.3.3 ([26]). A function $f(\cdot)$ defined over $\mathcal{X} \in \Re^n$ satisfies the diminishing returns (DR) property if for all $a \leqslant b \in \mathcal{X}$, for all $i \in n$, for all $k \in \Re_+$ such that $(k\mathcal{X}_i + a)$ and $(k\mathcal{X}_i + b)$ are still in $\mathcal{X}$, it holds, $f(k\mathcal{X}_i + a) - f(a) \geqslant f(k\mathcal{X}_i + b) - f(b)$. $f(\cdot)$ is called a DR-submodular function.

For a twice differentiable function Definition 5.3.3 is equivalent to $\nabla^2_{ij} f(x) \leqslant 0$, for all $i \neq j$ [18]. We introduce the following concepts which is used in Theorem 5.3.4 to prove that Problem 5.3.1 is equivalent to maximizing a DR-submodular function.

Notice that a strategy $p_{A_k} \in \mathbf{P}_{A_k}$ of the k^{th} adversary induces a set of sample paths in the underlying state space. All the sample paths resulting from a given p_{A_k} can be divided into one of the following two categories: i) a sample path, ω_k, starts from the node s_0 and ends at the node d_k in $\mathcal{G}$. Let $\Omega_k^{d_k}$ denote the set of such paths and ii) a sample path, ω_k, starts from the node s_0 and ends at a node $v_j \in V_{\mathcal{G}} \setminus \{d_k\}$, where $p_{A_k}(v_j, \varnothing) = 1$ and $j \in \{1, \ldots, N\}$. That is, $\mathcal{P}_{A_K}$ abandons its attack at a node v_j in $\mathcal{G}$ before reaching its destination d_k. Let Ω_k^{ϕ} denote the set of such paths.

Theorem 5.3.4. The defender's payoff function, $U_D(p_A, p_D)$, is DR-submodular in the defender's strategy, p_D.

Proof. For a given $p_A \in \mathbf{P}_A$, let $p_R(\omega_k)$, $p_T(\omega_k)$, and $p_\phi(\omega_k)$ be the probability of $\mathcal{P}_{A_K}$ reaching d_k without getting detected, the probability of getting detected, and the probability of getting trapped when k^{th} adversary is following the sample path ω_k, respectively. Define $p(\omega_k)$ to be the probability of $\mathcal{P}_{A_K}$ choosing sample path ω_k in $\mathcal{G}$. Now, $U_D(p_A, p_D)$ can be rewritten as follows.

$$\begin{aligned} U_D(p_A, p_D) = & \sum_{k \in \{1,\ldots,K\}} \Big(\sum_{\omega_k \in \Omega_k^{d_k}} p(\omega_k) \Big(p_T(\omega_k)\alpha_k^D + p_R(\omega_k)\beta_k^D + \\ & p_\phi(\omega_k)\sigma_k^D \Big) + \sum_{\omega_k \in \Omega_k^{\phi}} p(\omega_k) \Big(p_T(\omega_k)\alpha_k^D + p_\phi(\omega_k)\sigma_k^D \Big) \Big) \\ = & \sum_{k \in \{1,\ldots,K\}} \Big(\sum_{\omega_k \in \Omega_k^{d_k}} p(\omega_k) \Big(\alpha_k^D + [\beta_k^D - \alpha_k^D] p_R(\omega_k) + \\ & [\sigma_k^D - \alpha_k^D] p_\phi(\omega_k) \Big) + \sum_{\omega_k \in \Omega_k^{\phi}} p(\omega_k) \Big(\alpha_k^D + [\sigma_k^D - \alpha_k^D] p_\phi(\omega_k) \Big) \Big) \end{aligned} \tag{5.4}$$

Eq. (5.4) holds as $p_T(\omega_k) = 1 - p_R(\omega_k) - p_\phi(\omega_k)$. Notice that $p_R(\omega_k) = \prod_{s_i \in \omega_k} [1 - p_D^{\bar{s}}(s_i, k)]$ and $p_\phi(\omega_k) = \sum_{s_{i'} \in \omega_k} \prod_{s_i \in \hat{\omega}_k} [1 - p_D^{\bar{s}}(s_i, k)] p_{A_k}(s_{i'}, \varnothing)$, where $\hat{\omega}_k$ is the subpath in ω_k from the node s_0 to the node $s_{i'}$ in $\mathcal{G}$.

Let $\nabla^2_i U_D(p_D)$ denote the i^{th} entry of the gradient of U_D and $\nabla^2_{ij} U_D(p_D)$ denote the $(i,j)^{\text{th}}$

entry of Hessian of $U_D(p_D)$, where i and j are indices of the vector p_D. Then,

$$\nabla_i U_D(p_D) = \sum_{k\in\{1,\ldots,K\}} \Big(\sum_{\omega_k\in\Omega_k^{d_k}} p(\omega_k)\Big([\beta_k^D-\alpha_k^D]\nabla_i p_R(\omega_k) + [\sigma_k^D-\alpha_k^D]\nabla_i p_\phi(\omega_k)\Big) + \sum_{\omega_k\in\Omega_k^{\phi}} p(\omega_k)\Big([\sigma_k^D-\alpha_k^D]\nabla_i p_\phi(\omega_k)\Big)\Big)$$

$$\nabla^2_{ij} U_D(p_D) = \sum_{k\in\{1,\ldots,K\}} \Big(\sum_{\omega_k\in\Omega_k^{d_k}} p(\omega_k)\Big([\beta_k^D-\alpha_k^D]\nabla^2_{ij} p_R(\omega_k) + [\sigma_k^D-\alpha_k^D]\nabla^2_{ij} p_\phi(\omega_k)\Big) + \sum_{\omega_k\in\Omega_k^{\phi}} p(\omega_k)\Big([\sigma_k^D-\alpha_k^D]\nabla_{ij} p_\phi(\omega_k)\Big)\Big)$$

Here, $\nabla_i p_R(\omega_k) = -\prod_{\substack{s_l\in\omega_k\\ s_l\neq s_i}} [1-p_D^{\bar{s}}(s_l,k)]$ and for any $i\neq j$, $\nabla^2_{ij} p_R(\omega_k) = \prod_{\substack{s_l\in\omega_k\\ s_l\notin\{s_i,s_j\}}} [1-p_D^{\bar{s}}(s_l,k)]$. Also, for any $i\neq j$,

$\nabla_i p_\phi(\omega_k) = -\sum_{\substack{s_{l'}\in\omega_k\\ s_{l'}\geqslant s_i}} \prod_{\substack{s_l\in\hat{\omega}_k\\ s_l\neq s_i}} [1-p_D^{\bar{s}}(s_l,k)] p_{A_k}(s_{l'},\varnothing)$ and

$$\nabla^2_{ij} p_\phi(\omega_k) = \sum_{\substack{s_{l'}\in\omega_k\\ s_{l'}\geqslant\{s_i,s_j\}}} \prod_{\substack{s_l\in\hat{\omega}_k\\ s_l\notin\{s_i,s_j\}}} [1-p_D^{\bar{s}}(s_l,k)] p_{A_k}(s_{l'},\varnothing).$$

The notation $s_{l'}\geqslant s_i$ (and $s_{l'}\geqslant\{s_i,s_j\}$, resp.) implies that the node $s_{l'}$ in the sample path ω_k appears as or after the node s_i (s_r, where the index $r=i$ if the node s_i appears after the node s_j in the sample path ω_k and vice versa, resp.).

Note that $\nabla^2_{ij} p_R(\omega_k)\geqslant 0$ and $\nabla^2_{ij} p_\phi(\omega_k)\geqslant 0$. Then from Eq. (5.5), $\nabla^2_{ij} U_D(p_D)\leqslant 0$ for any $i\neq j$ as $[\beta_k^D-\alpha_k^D]\leqslant 0$ and $[\sigma_k^D-\alpha_k^D]\leq 0$. Thus, $U_D(p_D)$ is DR-submodular in p_D. □

In order to propose an approximation algorithm that leverages the DR-submodularity of U_D, we first show some additional properties of U_D. Specifically, in Lemma 5.3.5 we show that U_D is point-wise monotone increasing in p_D and in Theorem 5.3.6 we show a Lipschitz condition.

Lemma 5.3.5. The defender's payoff function, $U_D(p_A,p_D)$, is monotone increasing in the defender's strategy, p_D.

Proof. Consider the defender's payoff function given by Eq. (5.4). Notice that when p_D is increasing both $p_R(\omega_k) = \prod_{s_i \in \omega_k} [1 - p_D^{\bar{s}}(s_i, k)]$ and $p_\phi(\omega_k) = \sum_{s_{i'} \in \omega_k} \prod_{s_i \in \hat{\omega}_k} [1 - p_D^{\bar{s}}(s_i, k)] p_{A_k}(s_{i'}, \varnothing)$ terms decreases. Also $[\beta_k^D - \alpha_k^D] < 0$ and $[\sigma_k^D - \alpha_k^D] < 0$. Then for any $p_D, \hat{p}_D \in \mathbf{P}_D$ with $p_D < \hat{p}_D$, $U_D(p_A, p_D) < U_D(p_A, \hat{p}_D)$. □

A univariate auxiliary function $g_{p_D, \hat{p}_D}(\zeta)$ is introduced below to prove the required Lipschitz condition.

Theorem 5.3.6. The univariate axillary function $g_{p_D, \hat{p}_D}(\zeta) = U_D(p_A, p_D + \zeta \hat{p}_D)$ with respect to ζ has L-Lipschitz continuous derivative in $[0, 1]$, where $\zeta \in \mathbb{R}_+$ and $p_D, \hat{p}_D \in \mathbf{P}_D$.

Proof. Second derivative of $g_{p_D, \hat{p}_D}(\zeta)$ with respect to ζ can be written as follows:

$$\nabla_\zeta^2 g_{p_D, \hat{p}_D}(\zeta) = \sum_{k \in \{1, \ldots, K\}} \Big(\sum_{\omega_k \in \Omega_k^{d_k}} p(\omega_k) \Big([\beta_k^D - \alpha_k^D] \nabla_\zeta^2 p_R(\zeta, \omega_k) + [\sigma_k^D - \alpha_k^D] \nabla_\zeta^2 p_\phi(\zeta, \omega_k) \Big) + \sum_{\omega_k \in \Omega_k^\phi} p(\omega_k) [\sigma_k^D - \alpha_k^D] \nabla_\zeta^2 p_\phi(\zeta, \omega_k) \Big),$$

where $p_R(\zeta, \omega_k) = \prod_{s_i \in \omega_k} [1 - p_D^{\bar{s}}(s_i, k) - \zeta \hat{p}_D^{\bar{s}}(s_i, k)]$ and $p_\phi(\zeta, \omega_k) = \sum_{s_{i'} \in \omega_k} \prod_{s_i \in \hat{\omega}_k} [1 - p_D^{\bar{s}}(s_i, k) - \zeta \hat{p}_D^{\bar{s}}(s_i, k)] p_{A_k}(s_{i'}, \varnothing)$. Notice that both $p_R(\zeta, \omega_k)$ and $p_\phi(\zeta, \omega_k)$ are polynomials of ζ. Hence they can be expressed as $p_R(\zeta, \omega_k) = a_0 \zeta^n + a_1 \zeta^{n-1} + \ldots + a_{n-1} \zeta + 1$ and $p_\phi(\zeta, \omega_k) = b_0 \zeta^m + b_1 \zeta^{m-1} + \ldots + b_{m-1} \zeta + 1$. Furthermore, $0 \leq a_i, b_j \leq 1$ for $i \in \{0, \ldots, n-1\}$ and $j \in \{0, \ldots, m-1\}$, where n represent the number of distinct nodes (excluding the node corresponding to d_k) traversed in the sample paths related to $\omega_k \in \Omega_k^{d_k}$ and m is the number of distinct nodes traversed in $\omega_k \in \Omega_k^\phi$.

From the polynomial forms of $p_R(\zeta, \omega_k)$ and $p_\phi(\zeta, \omega_k)$, we can write their second derivatives with respect to ζ as $\nabla_\zeta^2 p_R(\zeta, \omega_k) = a_0 n(n-1) \zeta^{n-2} + a_1 (n-1)(n-2) \zeta^{n-3} + \ldots + 6 a_{n-3} \zeta + 2 a_{n-2}$ and $\nabla_\zeta^2 p_\phi(\zeta, \omega_k) = b_0 m(m-1) \zeta^{m-2} + b_1 (m-1)(m-2) \zeta^{m-3} + \ldots + 6 b_{m-3} \zeta + 2 b_{m-2}$. The terms a_i for $i \in \{0, \ldots, n-2\}$ and b_j for $j \in \{0, \ldots, m-2\}$ are products of probability terms. Hence, a_i and b_j are upper bounded by 1. $\max(n) = N - 1$ when the sample path from s_0 to d_k has to traverse through all the N distinct nodes in the $\mathcal{G}$ without getting trapped for a corresponding k^{th} analysis at $N - 1$ nodes (note that defender is not allowed to trap at the N^{th} node which is related to the d_k in this case). Similarly, $\max(m) = N - 1$ when the k^{th} adversary traverse through $N - 1$

distinct nodes in $\mathcal{G}$ on a path where N^{th} node is d_k (i.e $\omega_k \in \Omega_k^{d_k}$) and abandon the attack at the node $N-1$ with a non zero probability or k^{th} adversary traverse through $N-1$ distinct nodes in $\mathcal{G}$ that does not contain the node related to d_k and abandon the attack at node $N-1$ with probability one (i.e. $\omega_k \in \Omega_k^{\phi}$).

For $\zeta \in [0,1]$, we can bound the maximum value of $\nabla^2 p_R(\zeta, \omega_k)$ and $\nabla^2 p_\phi(\zeta, \omega_k)$ by $(N-2)(N-1)N/3$, where $N \geq 3$. Let ω'_k be a path in $\Omega_k^{d_k}$ that yield the highest probability $p(\omega'_k)$ and similarly let ω''_k be a path in Ω_k^{ϕ} that is gives the highest probability $p(\omega''_k)$. Let the total number of paths in the set $\Omega_k^{d_k}$ and Ω_k^{ϕ} denoted by $|\Omega_k^{d_k}|$ and $|\Omega_k^{\phi}|$, respectively. Define $\hat{L}_k = \max\{|p(\omega'_k)|\Omega_k^{d_k}|[\beta_k^D - \alpha_k^D]|, |p(\omega''_k)|\Omega_k^{d_k}|[\sigma_k^D - \alpha_k^D]|, |p(\omega''_k)|\Omega_k^{\phi}|[\sigma_k^D - \alpha_k^D]|\}$ and $\hat{L} = \max_k\{\hat{L}_k\}$ for $k = 1, \ldots, K$. Hence we obtain the following upper bound on the $|\nabla^2_\zeta g_{p_D, \hat{p}_D}(\zeta)|$:

$$|\nabla^2_\zeta g_{p_D, \hat{p}_D}(\zeta)| \leq \hat{L}_k \frac{(N-2)(N-1)N}{3} \leq \hat{L} \frac{(N-2)(N-1)N}{3}$$

Since the second derivative of $g_{p_D, \hat{p}_D}(\zeta)$ with respect to ζ is bounded in the case where $\zeta \in [0,1]$, first derivative of $g_{p_D, \hat{p}_D}(\zeta)$ with respect to ζ has L-Lipschitz continuous derivative in $[0,1]$. Furthermore, $L = \hat{L}\frac{(N-2)(N-1)N}{3}$. □

The Lipschitz constant L derived in Theorem 5.3.6 is used later in Algorithm 5 and in Proposition 5.3.9 to claim an approximation guarantee on Problem 5.3.1.

Definition 5.3.7. A set $\mathcal{P}$ is said to be a down-closed convex set if $x \in \mathcal{P}$ and $0 \leqslant y \leqslant x$ implies $y \in \mathcal{P}$.

Lemma 5.3.8. Let $\mathbf{P}_D$ be the strategy space of the defender and $\mathcal{P} \subset \mathbf{P}_D$ be the feasible strategy space of the defender. That is, $\mathcal{P} := \{p_D \in \mathbf{P}_D : \sum_{i=1}^{W} \sum_{k=1}^{K} p_D^{\bar{s}}(s_i, k)\mathcal{C}_k(s_i) \leqslant M, \text{for all } \bar{s} \in \bar{\mathcal{S}}\}$. Then, $\mathcal{P}$ is a down-closed convex polytope in the positive orthant.

Proof. The strategy space of the defender is a positive orthant as probabilities are bounded between 0 and 1. Thus $\mathcal{P} \subseteq [0,1]^{|p_D|}$ satisfies $\mathcal{P} := \{p_D \in \mathbf{P}_D | 0 \leqslant [0,1]^{|p_D|}, Ap_D \leqslant M\mathbb{1}\}$, where $\mathbb{1}$ is a vector of all ones of size $|\bar{\mathcal{S}}|$. Here, $|\bar{\mathcal{S}}|$ is the number of states in $\bar{\mathcal{S}}$, $|p_D|$ is the size of p_D, M is the available memory, and A is the $(|\bar{\mathcal{S}}| \times |\bar{\mathcal{S}}|)$ matrix that captures the memory constraint. Note that the feasible space is a subset of the polytope $[0,1]^{|p_D|}$ that is constrained by $Ap_D \leqslant M\mathbb{1}$. For any

$\hat{p}_D$ satisfying $0 \leqslant \hat{p}_D \leqslant p_D$, $A\hat{p}_D \leqslant M\mathbb{1}$. This implies that $\mathcal{P}$ is a down-closed convex polytope in the positive orthant. □

In what follows, we present an algorithm to compute an approximate optimal strategy of the defender.

Algorithm 5 Best response computation of the defender

Input: Attacker's strategies $\mathcal{P}_A$, $\mathcal{P}$, stepsize $\gamma \in (0, 1]$

Output: Best response of the defender $\bar{p}_D$

1: Initialize $p_D^0 \leftarrow 0, t \leftarrow 0, r \leftarrow 0$

2: **while** $t < 1$ **do**

3: Find $\hat{p}_D^r : \langle \hat{p}_D^r, \nabla U_D(p_D^r) \rangle \geqslant \eta \max_{\hat{p}_D \in \mathcal{P}} \langle \hat{p}_D, \nabla U_D(p_D^r) \rangle - \frac{1}{2}\delta L$, where $L > 0$ is the Lipschitz constant from Theorem 5.3.6, $\eta \in (0, 1]$ is the multiplicative error level, $\delta \in [0, \bar{\delta}]$ is the additive error level

4: Find stepsize $\gamma_r \in (0, 1]$, e.g., $\gamma_r \leftarrow \gamma$; and set $\gamma_r \leftarrow \min\{\gamma_r, 1 - t\}$

5: $p_D^{r+1} \leftarrow p_D^r + \gamma_r \hat{p}_D^r$, $t \leftarrow t + \gamma_r$, $r \leftarrow r + 1$

6: **end whilereturn** $\bar{p}_D \leftarrow p_D^r$

Proposition 5.3.9. Let $U_D^\star(p_A)$ be the maximum defender's payoff for given attackers' strategies $p_A = \{p_{A_k}\}_{k=1}^K$. Then, Algorithm 5 which takes as input p_A returns an approximate optimal defense strategy $\bar{p}_D$ for the defender such that $U_D(p_A, \bar{p}_D) \leqslant (1 - 1/e + \epsilon) U_D^\star(p_A)$. Further, Algorithm 5 takes $O(1/\epsilon)$ number of iterations and the number of operations in each iteration is linear in the action space of the defender.

Proof. Theorem 5.3.4 and Lemma 5.3.5 show that the defender's payoff function, U_D, is DR-submodular and monotonically increasing in p_D, respectively. Thus Problem 5.3.1 is equivalent to maximizing an increasing DR-submodular function. Using the Lipschitz constant of the gradient of U_D derived in Theorem 5.3.6, the proof follows from Theorem 1 and Corollary 1 in [26]. □

5.3.2 *Best Responses of the Adversaries*

The best response of the k^{th} adversarial player $\mathcal{P}_{A_k}$ to a given defender strategy, p_D, and other $K-1$ adversarial players' strategies, $p_{A\setminus k} := \{p_{A_k}\}_{k=1}^{k=K} \setminus p_{A_k} = p_{A_1}, \ldots, p_{A_{k-1}}, p_{A_{k+1}}, \ldots, p_{A_K}$, is given by the following optimization problem.

Problem 5.3.10. *The adversary's problem is as follows:* for any $k \in \{1, \ldots, K\}$

$$\max_{p_{A_k} \in \mathbf{P}_{A_k}} \quad \Big(\bar{p}_T(k)\alpha_k^A + \bar{p}_R(k)\beta_k^A + \bar{p}_\phi(k)\sigma_k^A\Big)$$

In this section we derive a pure strategy best response for the k^{th} adversary, where $k \in \{1, \ldots, K\}$.

Definition 5.3.11 (Pure strategies of $\mathcal{P}_{A_k}$). The k^{th} adversary's strategy can be interpreted as selecting a sample path, ω_k, in $\mathcal{G}$ from node s_0 to d_k. Then for a given pair of nodes v_i and v_j in $\mathcal{G}$, where $i, j \in \{1, \ldots, N\}$, a pure strategy of $\mathcal{P}_{A_k}$ can be defined as follows:

$$p_{A_k}(v_i, v_j) = \begin{cases} 1, & \text{if } (v_i, v_j) \in \omega_k \\ 0, & \text{otherwise.} \end{cases}$$

The following lemma characterizes the best response of $\mathcal{P}_{A_k}$ under pure strategies.

Lemma 5.3.12. Let $\Omega_k^{d_k}$ denote the set of paths that start from the node s_0 and end at the node d_k in $\mathcal{G}$. Then for any $\omega_k \in \Omega_k^{d_k}$, define $p_R(\omega_k)$ to be the probability of $\mathcal{P}_{A_k}$ reaching d_k without being detected by $\mathcal{P}_D$ when $\mathcal{P}_{A_k}$ is following the path ω_k. Furthermore, let $\omega^* = \arg\max\{p_R(\omega_k) : \omega_k \in \Omega_k^{d_k}\}$. Then, $\omega^* \in \text{BR}(p_D, p_{A\setminus k})$.

Proof. Note that $p_{A_k}(v_i, \phi) = 0$, for all $i \in \{1, \ldots, N\}$, when $\mathcal{P}_{A_k}$ follows a pure strategy. Let $p(\omega_k)$ be the probability of $\mathcal{P}_{A_K}$ choosing a path $\omega_k \in \Omega_k^{d_k}$. Then the payoff of $\mathcal{P}_{A_k}$ under a pure strategy is as follows:

$$\begin{aligned} U_{A_k}(p_A, p_D) &= \bar{p}_T(k)\alpha_k^A + \bar{p}_R(k)\beta_k^A \\ &= \sum_{\omega_k \in \Omega_k^{d_k}} p(\omega_k)\Big([1 - p_R(\omega_k)]\alpha_k^A + p_R(\omega_k)\beta_k^A\Big) \\ &= \sum_{\omega_k \in \Omega_k^{d_k}} p(\omega_k)\Big(\alpha_k^A + [\beta_k^A - \alpha_k^A]p_R(\omega_k)\Big) \end{aligned}$$

For a given p_D and $p_{A\setminus k}$, any path $\omega_k \in \Omega_k^{d_k}$ that maximize $U_{A_k}(p_A, p_D)$ will yield the same payoff to $\mathcal{P}_{A_k}$. Hence we can rewrite the best response of $\mathcal{P}_{A_k}$ (under pure strategies) as

$$\text{BR}(p_D, p_{A\setminus k}) \in \arg\max_{\omega_k \in \Omega_k^{d_k}} \Big(\alpha_k^A + [\beta_k^A - \alpha_k^A] p_R(\omega_k)\Big).$$

Since α_k^A is a constant value and the term $[\beta_k^A - \alpha_k^A]$ is a positive scalar, $\text{BR}(p_D, p_{A\setminus k}) \in \arg\max\{p_R(\omega_k) : \omega_k \in \Omega_k^{d_k}\}$. □

Next we define the set of states reachable to a state $\bar{s} \in \bar{\mathcal{S}}$ in the following definition.

Definition 5.3.13. A state $\bar{s} \in \bar{\mathcal{S}}$ is said to be *one-step reachable* from a state $\bar{\bar{s}} \in \bar{\mathcal{S}}$ if each of the position s_j in the state $\bar{s}$ is one-step reachable from each of the position $\bar{s}_j$ in $\bar{\bar{s}}$, for all $j \in \{1, \ldots, W\}$. Here, we say a position s_j is one-step reachable from a position $\bar{s}_j$, if $s_j \in \{\phi, \tau\}$ or if $(v_i, v_r) \in E_{\mathcal{G}}$ where $\bar{s}_j = v_i$ and $s_j = v_r$. Then, $\bar{\mathbf{S}}(\bar{s}) \subseteq \bar{\mathcal{S}}$ is the set of states from which state $\bar{s}$ is one-step reachable.

Let p_B be the distribution of the benign information flows in the system. More precisely, it provides transition probabilities of a benign flow between two nodes $v_i, v_j \in V_{\mathcal{G}}$. In the following, we define the probability of type-k trapping at node v_i in $\mathcal{G}$, $p_D(v_i, k)$, for a given p_D, $p_{A\setminus k}$, and p_B.

Definition 5.3.14. Without loss of generality, assume each k^{th} position in the state $\bar{s} \in \bar{\mathcal{S}}$ related to the $\mathcal{P}_{A_k}$. Let $\hat{\mathbf{S}}(v_i) \subseteq \bar{\mathcal{S}}$ be the set of states such that for all $\hat{\mathbf{s}} \in \hat{\mathbf{S}}(v_i)$, $s_k = v_i$ for $i \in \{1, \ldots, N\}$. Then the probability of type-k trapping at node v_i in $\mathcal{G}$, $p_D(v_i, k)$, can be defined as follows:

$$p_D(v_i, k) = \sum_{\hat{\bar{s}} \in \hat{\mathbf{S}}(v_i)} \Big(\sum_{\bar{\bar{s}} \in \bar{\mathbf{S}}(\hat{\mathbf{s}})} \prod_{k'} p_{A_{k'}}(\bar{s}_{k'}, \hat{s}_{k'}) \prod_{k''} p_B(\bar{s}_{k''}, \hat{s}_{k''}) \Big) p_D^{\hat{\bar{s}}}(\hat{s}_k, k),$$

where $k' \in \{1, \ldots, K\} \setminus k$, $k'' \in \{K+1, \ldots, W\}$,

$$p_{A_{k'}}(\bar{s}_{k'}, \hat{s}_{k'}) = \begin{cases} 1, \text{ if } \bar{s}_{k'} = \hat{s}_{k'} \text{ and } \bar{s}_{k'} \in \{\phi, \tau, d_{k'}\} \\ p_D^{\bar{\bar{s}}}(\bar{s}_{k'}, k'), \text{ if } \bar{s}_{k'} \in V_{\mathcal{G}} \text{ and } \hat{s}_{k'} = \tau \\ p_{A_{k'}}(\bar{s}_{k'}, \hat{s}_{k'})[1 - p_D^{\bar{\bar{s}}}(\bar{s}_{k'}, k')], \text{ otherwise} \end{cases}$$

and

$$p_B(\bar{s}_{k''}, \hat{s}_{k''}) = \begin{cases} 1, \text{ if } \bar{s}_{k''} = \hat{s}_{k''} \text{ and } \bar{s}_{k''} \in \{\phi, d_{\bar{k}}\} \\ p_B(\bar{s}_{k''}, \hat{s}_{k''}), \text{ otherwise} \end{cases}$$

for all $\bar{k} \in \{1, \ldots, K\}$.

The following theorem presents a method to calculate a best response of $\mathcal{P}_{A_k}$ for a given p_D, $p_{A\setminus k}$, and p_B.

Theorem 5.3.15. The best response of adversary $\mathcal{P}_{A_k}$, BR$(p_D, p_{A\setminus k})$, under pure strategies, is a path ω^* returned by a shortest path algorithm on the IFG, $\mathcal{G}$, with edge weight of each incoming edge to a node v_i given by $-\log([1 - p_D(v_i, k)])$, for $i \in \{1, \ldots, N\}$.

Proof. Notice that using $p_D(v_i, k)$ we can write $p_R(\omega_k) = \prod_{v_i \in \omega_k} [1 - p_D(v_i, k)]$. From Lemma 5.3.12, we can derive the following expression[1] for the payoff of $\mathcal{P}_{A_k}$ under best response strategy,

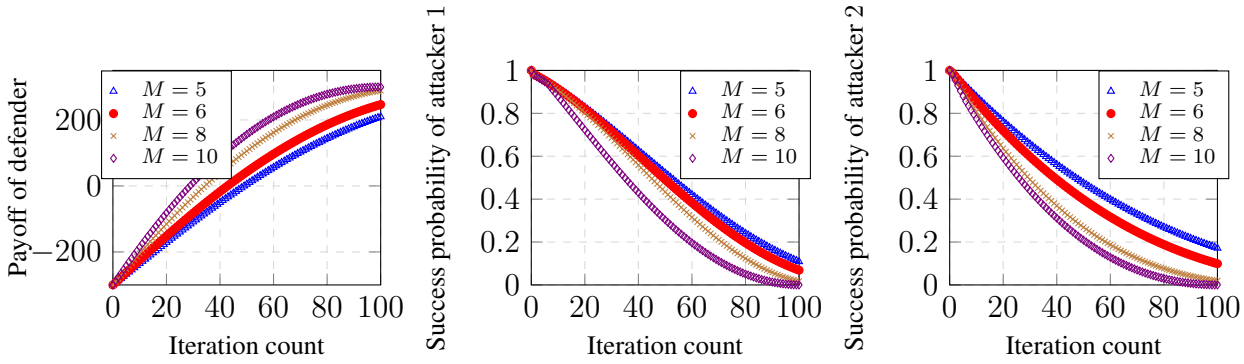
$$\begin{aligned} \max \prod_{v_i \in \omega_k} [1 - p_D(v_i, k)] &= \max \prod_{v_i \in \omega_k} \log([1 - p_D(v_i, k)]) \\ &= \min \prod_{v_i \in \omega_k} -\log([1 - p_D(v_i, k)]) \end{aligned}$$

This implies that solving Problem 5.3.10 is equivalent to solving a shortest path algorithm on $\mathcal{G}$, with $-\log([1 - p_D(v_i, k)])$ as the edge weight of each incoming edge to node v_i. Notice that edge weights are positive values since $0 < [1 - p_D(v_i, k)] \leqslant 1$ Furthermore, Definition 5.3.14 can be used to calculate $p_D(v_i, k)$ for given p_D, $p_{A\setminus k}$ and p_B. Hence a Dijkstra's shortest path algorithm [43] will compute an optimal attacker strategy. □

5.4 Simulation Study

We validate our theoretical results using real-world attack data obtained using RAIN [64] for a three day nation state attack. We implement our model and run Algorithm 5 on day one attack data of the nation state. A brief description of the dataset we used and the steps involved in the construction of the IFG for that attack are given below.

[1]The expression gives a scaled version of the exact payoff. In order to calculate the exact payoff under best responses of $\mathcal{P}_{A_k}$, one need to multiply the value of this expression by $[\beta_k^A - \alpha_k^A]$ and add a constant value α_k^A.



(a) **(b)** **(c)**

Figure 5.1: The parameters chosen are: $\alpha_1^D = 100$, $\alpha_2^D = 200$, $\beta_1^D = -100$, and $\beta_1^D = -200$. The memory values, $\mathcal{C}_k$, for the nodes in the IFG where chosen from a random distribution such that $\mathcal{C}_1(v_i) < \mathcal{C}_2(v_i)$, for all $i \in \{1, \ldots, N\}$. Figure 1(a) shows the payoff of the defender obtained by Algorithm 5 v.s. iteration count for four instances with different values of M. Figures 1(b) and 1(c) show the probability of attacker one and two, respectively, reaching the target with increasing values of M.

The goal of the adversaries' campaign was to steal sensitive proprietary and personal information from the targeted company. We considered two attackers operating in the system with distinct goals and entry points. Both attackers established their initial foothold in the system through network (e.g., spear-phishing attack, watering hole attack). Once the system is compromised, attacker 1 leveraged common system utilities to perform internal reconnaissance. The goal of this attacker was to fingerprint the compromised system to detect running processes and network information. On the other hand, attacker 2 wrote a malicious program to a disk that was eventually executed to establish a backdoor which is used to continuously exfiltrate the company's sensitive data.

Initial conversion of the attack data into an IFG resulted in a coarse-grain graph with $\approx$ 132,000 nodes and $\approx$ 2 million edges. We pruned the coarse-grained graph by starting from the destinations of each attacker. Then we constructed the subgraph related to all the nodes in the coarse-grained graph that have at least one directed path to the destination of that attacker. We performed our analysis on the resulting refined IFG consisting of 30 nodes related to the attacks.

The parameters chosen are: $\alpha_1^D = 100$, $\alpha_2^D = 200$, $\beta_1^D = -100$, and $\beta_1^D = -200$. Thus

attacker 2 is more capable as the impact of attack 2 is more compared to attack 1, which is captured by $\alpha_2^D > \alpha_1^D$. Figure 1(a) shows the variation of the payoff of the defender returned by Algorithm 5 with respect to iteration count for four instances with different values of M. In order to analyze the impact of the defender on the different attackers, we compute attack success probabilities of each attacker with varying the amount of memory. Figures 1(b) and 1(c) show the probabilities of attacker one and two, respectively, reaching their targets while increasing the value of M. Notice from Figures 1(b) and 1(c) that for a fixed memory constraint, attacker 2 has higher probability of reaching the target compared to attacker 1 which is expected as attacker 2 is more capable.

5.5 Summary

In this chapter, we studied the problem of detecting multiple attackers in a computer system. We presented an analytical model of a resource constrained DIFT that allocate scarce resources across multiple flows to simultaneously detect different attackers. We modeled the strategic interaction between K adversarial information flows and the DIFT defense as a dynamic $(K + 1)$-player game. Each stage of the game corresponds to the propagation of the attacks through the system, in which each attacker must determine the next operation and the defender must decide an efficient memory allocation. Given attackers' strategies, we proved that finding an optimal defense strategy is equivalent to maximizing an increasing DR-submodular function which enabled us to propose an approximation algorithm. Further, given a defense strategy and strategies of $(K - 1)$ attackers, we showed that finding an optimal attacker strategy is equivalent to solving a shortest path problem, where the edge weights are derived from the strategies of the other players. Based on this mapping we proposed a polynomial-time algorithm for computing an optimal attacker strategy. We evaluated the performance of our algorithm on a real-world attack dataset obtained using RAIN [64].

Chapter 6

TOWARDS STOCHASTIC DIFT-APT GAMES

6.1 Motivation

Information flows consists of explicit information flows arising from data dependencies and implicit information flows that arise from control dependencies [97]. *Conditional branches* occur in the program due to control dependencies. During the initial phase of the attack, as the system is not fully compromised and APTs undergo system exploration, the traversal of the APTs through the system is controlled not only by the actions of APTs but also by the dependencies of variables or processes in the system at the conditional branches in the program. For instance, consider a function in the program that has an *if-else* statement such that a variable, say θ, decides whether the *if*-part or the *else*-part of the code get executed. The value of θ and hence the attack path is affected by both benign and malicious flows in the system and not solely by the actions of the APT. Conditional branching introduces randomness in the attack paths of APTs. Most of the existing architectures of DIFT have been developed to track only data-flow-based information flows and hence captures only a portion of the complete program behavior [120]. Further, current approaches of using DIFT often introduces performance and memory overhead and are expensive.

Our objective is to obtain an analytical model of DIFT that can monitor the system online in a resource-efficient manner against APTs by considering both data-flow dependencies and conditional branches. The contributions of this chapter are as follows:

• We model the interaction between APTs and DIFT as a two-player, nonzero-sum *stochastic game* in infinite-horizon. The stochastic nature of the game captures the uncertainty involved at the conditional branches. The state space of the game and the action sets of both the players are finite. Further, the players strategize from a stochastic policy to maximize their individual payoffs.

• We relate the best response of the APTs to a *maximum reachability probability problem* and

formulate it as a linear program. We formulate the best response of the DIFT as a *linear program* and present a polynomial-time algorithm to calculate a deterministic optimal policy of the DIFT.

• As deriving Nash equilibrium for nonzero-sum infinite-horizon stochastic games is computationally difficult, we formulate discounted version of the game to perform approximate equilibrium analysis. Then we present a value-iteration algorithm with guaranteed convergence and prove that the algorithm returns an ϵ-Nash equilibrium for the undiscounted game in polynomial-time.

• To evaluate the performance of our approach, we perform experimental analysis of the model and the algorithm on augmented version of NetRecon data obtained from Refinable Attack INvestigation (RAIN) [64] framework.

6.2 Stochastic DIFT-APT Game Formulation

In this section, we model a two-player game between APTs ($\mathcal{P}_A$) and DIFT ($\mathcal{P}_D$) on the information flow graph $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$. The objective of $\mathcal{P}_A$ is to reach critical node set $\mathcal{D} \subset V_\mathcal{G}$ and the objective of $\mathcal{P}_D$ is to perform security analysis at optimal locations in $\mathcal{G}$ so as to prevent $\mathcal{P}_A$ from reaching $\mathcal{D}$. The game unfolds on a finite state space $\mathbf{S} := V_\mathcal{G} \cup \{\phi, \tau\} = \{s_1, \ldots, s_N, \phi, \tau\}$. State ϕ is a null state that is reached when the adversarial flow is dropped. State τ is a trap state that is reached when the defense traps a flow. Notice that ϕ, τ are terminal or absorbing states of the state space $\mathbf{S}$. We denote the *state* of the game at time $t \in \mathcal{T} := \{1, 2, \ldots\}$ by the random variable $\bar{s}_t$. Due to conditional branching in the program, the transitions from a current state $\bar{s}_t$ to a next state $\bar{s}_{t+1}$ in $\mathbf{S}$ is determined over a probability distribution, and hence the game is stochastic.

6.2.1 Transition Probability

The action space of $\mathcal{P}_D$ and $\mathcal{P}_A$ are denoted by the sets $\mathcal{A}_D$ and $\mathcal{A}_A$, respectively. At time t, players $\mathcal{P}_D$ and $\mathcal{P}_A$ affects the game through their actions $d_t \in \mathcal{A}_D$ and $a_t \in \mathcal{A}_A$, respectively. As states ϕ, τ are absorbing states of $\mathbf{S}$, the action sets of both the players at these states are empty, i.e., $\mathcal{A}_A(\bar{s}_t) = \mathcal{A}_D(\bar{s}_t) = \emptyset$ for $\bar{s}_t \in \{\phi, \tau\}$. Now we define the action set of the players at a state $\bar{s}_t \notin \{\phi, \tau\}$. For each node in the information flow graph, say $s_i \in V_\mathcal{G}$, let $\mathcal{N}(s_i) := \{s_j : (s_i, s_j) \in E_\mathcal{G}\}$

denote the neighbor set of s_i. Then the action sets of the players at state $\bar{s}_t \notin \{\phi, \tau\}$ are given by $\mathcal{A}_{A}(\bar{s}_t) = \{\mathcal{N}(s_i)\} \cup \{\text{quit}\}$ and $\mathcal{A}_{D}(\bar{s}_r) = \{1, 0\}$. More precisely, at a state $\bar{s}_t \notin \{\phi, \tau\}$, the adversary decides to transition to one of the neighboring process of s_i or quit the game and the defender decides to trap the information flow or not. The stationary transition probability from a given state s to a state s' under actions $a \in \mathcal{A}_{A}$ and $d \in \mathcal{A}_{D}$ is

$$p(s'|s, a, d) := \mathbb{P}\{\bar{s}_{t+1} = s'|\bar{s}_t = s, a_t = a, d_t = d\}, \tag{6.1}$$

for all $s, s' \in \mathbf{S}$, $a \in \mathcal{A}_{A}$ and $d \in \mathcal{A}_{D}$, where $p : \mathbf{S} \times \mathbf{S} \times \mathcal{A}_{A} \times \mathcal{A}_{D} \to [0, 1]$. Note that, given $\bar{s}_t, a_t$, and d_t, the state $\bar{s}_{t+1}$ is assumed to be conditionally independent of all other random variables at time less than or equal to t. Here, $p(s'|s, a, d) \neq 1$ for some states $s, s' \in \mathbf{S}$ due to the conditional branching that occurs at the subset of processes.

6.2.2 Information of the Game

The players $\mathcal{P}_{D}$ and $\mathcal{P}_{A}$ decide their actions based on present state of the game. We denote the information available to the players $\mathcal{P}_{D}$ and $\mathcal{P}_{A}$ at time $t \in \mathcal{T}$ by $\mathbf{Y}_t$ and $\mathbf{Z}_t$, respectively. Then

$$\begin{aligned} \mathbf{Y}_t &:= \{\bar{s}_0, d_0, \bar{s}_1, d_1, \ldots, \bar{s}_{t-1}, d_{t-1}, \bar{s}_t\}, \\ \mathbf{Z}_t &:= \{\bar{s}_0, a_0, \bar{s}_1, a_1, \ldots, \bar{s}_{t-1}, a_{t-1}, \bar{s}_t\}. \end{aligned}$$

We denote the set of all possible outcomes for $\mathbf{Y}_t$ and $\mathbf{Z}_t$ at time t using $\mathcal{Y}^\star$ and $\mathcal{Z}^\star$, respectively. At every time step in the game, the defense decides whether to trap the flow or not and the adversary decides which neighboring node to transition. While both the players know the current state of the game, the players do not know the action of the opponent player at a state. Moreover, the defense is unaware whether a tagged (suspicious) flow is malicious and the adversary does not know the probability of getting trapped at different processes in the system. This results in information *asymmetry* among the players.

6.2.3 Stochastic Player Policies

At any time $t \in \mathcal{T}$, $\mathcal{P}_D$ and $\mathcal{P}_A$ selects an action from the action set $\mathcal{A}_D$ and $\mathcal{A}_A$, respectively, based on some probability distribution. A stochastic policy of player $\mathcal{P}_D$ is given by $p_D : \mathcal{Y}^\star \rightarrow [0,1]^{\mathcal{A}_D}$ and of player $\mathcal{P}_A$ is given by $p_A : \mathcal{Z}^\star \rightarrow [0,1]^{\mathcal{A}_A}$. The player policies we consider here are *stochastic* and *stationary* in nature. Since the policy is stationary, the probability distribution at any $\mathbf{Y}_t$ or $\mathbf{Z}_t$ only depends on the current state $\bar{s}_t$. The set of all stochastic policies of $\mathcal{P}_D$ and $\mathcal{P}_A$ are given by $\mathbf{p}_D$ and $\mathbf{p}_A$, respectively. A policy $p_D \in \mathbf{p}_D$ ($p_A \in \mathbf{p}_A$) is said to be a *pure* or *deterministic* policy if for all $Y \in \mathcal{Y}^\star$ ($Z \in \mathcal{Z}^\star$), the entries of the vector $p_D(Y)$ ($p_A(Z)$) belong to the set $\{0,1\}$.

6.2.4 Payoffs to the Players

The payoff functions of the players are denoted by U_D and U_A. The game is non-cooperative and both the players try to maximize their own payoff. The payoff function of the defense consists of four components (i) a memory cost $\mathcal{C}_D(s_i) < 0$ for selecting $s_i \in V_\mathcal{G}$ as a security check location, i.e., tag sink, (ii) penalty $\beta^D < 0$ for adversary reaching a destination, (iii) reward $\alpha^D > 0$ for defense catching the adversary and (iv) reward $\sigma_D > 0$ for adversary dropping out of the game. Similarly, the payoff function of the adversary consists of three components (i) reward $\beta^A > 0$ for adversary reaching a destination, (ii) penalty $\alpha^A < 0$ if the adversary is caught by the defense and (iii) penalty $\sigma_A < 0$ for adversary dropping out of the game. Although all tagged flows are suspicious, most of them are often benign. The defense cannot distinguish a malicious data and spurious data from tagged flows. The trapping cost $\mathcal{C}_D(s)$ captures the cost of trapping benign flows at process s in the system.

Given set of policies (p_D, p_A), where $p_D \in \mathbf{p}_D$ and $p_A \in \mathbf{p}_A$, the payoff functions of the players are given by

$$U_D(p_D, p_A) = \sum_{s_i \in \mathcal{S}} \Big(p_D(s_i) \mathcal{C}_D(s_i) \Big) + p_\mathcal{T}\, \alpha^D + p_R\, \beta^D + p_\phi\, \sigma_D, \tag{6.2}$$

$$U_A(p_D, p_A) = p_\mathcal{T}\, \alpha^A + p_R\, \beta^A + p_\phi\, \sigma_A, \tag{6.3}$$

where $p_{\mathcal{T}}$ is the probability that the adversarial flow is caught by the defense, p_R is the probability that the adversarial flow reaches a destination and p_ϕ is the probability that the adversary drops out from the game ($p_\phi = 1 - p_{\mathcal{T}} - p_R$), under policies (p_D, p_A). In Eq. (6.2), the first term corresponds to the resource cost to the defender. The second, third, and fourth terms in Eq. (6.2) model the reward/penalty the defender incurs for winning the game, losing the game, and a partial reward if adversary abandons the attack, respectively. Similarly, the first, second, and third terms in Eq. (6.3) capture the penalty/reward the adversary incurs for losing the game, winning the game, and the partial penalty for abandoning the attack, respectively.

6.3 Solution to Stochastic DIFT-APT Game

In this section, we first presents an overview of the notion of equilibrium considered in this paper. We then analyze the best responses of both the players and propose a method to solve for the Nash equilibrium of the game.

We present the concept of a player's best response to a given mixed strategy of the opponent below.

Definition 6.3.1. Let $p_D : \mathcal{Y}^\star \to [0,1]^{\mathcal{A}_D}$ denote a defender strategy and $p_A : \mathcal{Z}^\star \to [0,1]^{\mathcal{A}_A}$ denote an adversary strategy. The set of best responses of the defender is given by $\text{BR}(p_A) = \arg\max_{p_D}\{U_D(p_D, p_A) : p_D \in [0,1]^{\mathcal{A}_D}\}$. Similarly, the best responses of the adversary given by $\text{BR}(p_D) = \arg\max_{p_A}\{U_A(p_D, p_A) : p_A \in [0,1]^{\mathcal{A}_A}\}$.

A mixed strategy profile (p_D, p_A), where $p_D \in \mathbf{p}_D$ and $p_A \in \mathbf{p}_A$, is a *Nash equilibrium* (NE) if the following definition hold.

Definition 6.3.2. A pair of mixed strategies $(p_D^\star, p_A^\star)$ is a *Nash equilibrium* if

$$p_D^\star \in \text{BR}(p_A^\star) \text{ and } \quad p_A^\star \in \text{BR}(p_D^\star).$$

Definition 6.3.2 translates to players selecting policies $(p_D^\star, p_A^\star) \in \mathbf{p}_D \times \mathbf{p}_A$ such that

$$U_D(p_D^\star, p_A^\star) \;\geqslant\; U_D(p_D, p_A^\star), \text{ for all } p_D \in \mathbf{p}_D \text{ and,} \tag{6.4}$$

$$U_A(p_D^\star, p_A^\star) \;\geqslant\; U_A(p_D^\star, p_A), \text{ for all } p_A \in \mathbf{p}_A. \tag{6.5}$$

The policies $(p_D^\star, p_A^\star)$ satisfying Eqs.(6.4) and (6.5) are said to constitute a Nash equilibrium in the stochastic policy. Now we define the notion of ϵ-Nash equilibrium here.

Definition 6.3.3. $(p_A^\epsilon, p_D^\epsilon) \in \mathbf{p}_A \times \mathbf{p}_D$ forms an ϵ-equilibrium in stochastic stationary strategies for any $\epsilon > 0$ and for all $p_A \in \mathbf{p}_A$ and $p_D \in \mathbf{p}_D$ if

$$\begin{aligned} U_A(p_A^\epsilon, p_D^\epsilon) &\geqslant U_A(p_A, p_D^\epsilon) - \epsilon, \text{ and} \\ U_D(p_A^\epsilon, p_D^\epsilon) &\geqslant U_D(p_A^\epsilon, p_D) - \epsilon. \end{aligned}$$

Although the existing literature on stochastic games ensures existence of Nash equilibrium for a finite-horizon stochastic game under stochastic behavioral policies, proving existence and deriving a Nash equilibrium for a nonzero-sum, undiscounted, infinite-horizon stochastic game is computationally difficult. In the next section we propose methods to calculate best responses of the players and to derive an approximate Nash equilibrium.

6.3.1 Best Response of the Adversary

For a given defender's policy the adversary's optimal strategy is to choose an action at every state in the game that will maximize the probability of adversary reaching some node in $\mathcal{D}$. We now prove that the best response calculation of the adversary reduces to a *maximum probability reachability problem* [20] with reachability set $\mathcal{D}$. We introduce the following concepts before stating the result. The set $\mathcal{D}$ is said to be reachable from state $s \in \mathbf{S}$ denoted by $s \rightsquigarrow \mathcal{D}$ if there exists a directed path from s to some node in $\mathcal{D}$. Consider the following Markov Decision Process (MDP) problem.

Problem 6.3.4. Let p_D denote the defender's policy. For all s satisfying $s \notin \mathcal{D}$ and $s \rightsquigarrow \mathcal{D}$, find

$x_s = \max\limits_{a \in \mathcal{A}_A(s)} \{ \sum\limits_{s' \in \mathbf{S}} p_D(s', 0)\, p(s'|s, a, 0)\, x_{s'} \}$

Subject to:

(i) $x_s = 1$ *if* $s \in \mathcal{D}$*, and*

(ii) $x_s = 0$ *if* $\mathcal{D}$ *is not reachable from* s*.*

Now we state and prove the result showing relation between best response of the adversary and the maximal probability reachability problem given in Problem 6.3.4.

Theorem 6.3.5. Let

$$a^\star(s) \in \arg\max_{a \in \mathcal{A}_A(s)} \{ \sum_{s' \in \mathbf{S}} p_D(s', 0)\, p(s'|s, a, 0)\, x_{s'} \}.$$

Then, $p_A^\star$ defined as $p_A^\star(s) := a^\star(s)$ is a best response for the adversary.

Proof. Note that $p_D(s', 0)$ denotes the probability that information flow is not trapped at a process $s' \in \mathbf{S}$. Hence, vector $(x_s)_{s \in \mathbf{S}}$ is the maximum probability that an adversarial flow originating at state $s \in \mathbf{S}$ reach a state in $\mathcal{D}$ without getting trapped by the defense for the given defense policy p_D. By Eq. (6.3), at any state $s \in \mathbf{S}$ the best action of the adversary is to choose action that maximizes its probability of reaching $\mathcal{D}$, given by $\arg\max_{a \in \mathcal{A}_A(s)} \Big(\sum_{s' \in \mathbf{S}} p_D(s', 0)\, p(s'|s, a, 0)\, x_{s'} \Big)$. Thus the proof follows. □

As a consequence of Theorem 6.3.5, one can calculate the best response of the adversary for any given defense policy by solving Problem 6.3.4. Using the approach in [20], we now show in Theorem 6.3.6 that Problem 6.3.4 reduces to a constrained linear optimization problem.

Theorem 6.3.6. Let $(x_s)_{s \in \mathbf{S}}$ be the vector such that x_s is the maximum probability that an adversarial flow originating at state $s \in \mathbf{S}$ reach a state in $\mathcal{D}$ for the given defense policy p_D. Then $(x_s)_{s \in \mathbf{S}}$ is the unique solution to the following linear program (LP).

For $s \notin \mathcal{D}$ and $s \rightsquigarrow \mathcal{D}$ and for all $a \in \mathcal{A}_A(s)$: find

$y_s \geqslant \sum_{s' \in \mathbf{S}} p_D(s', 0)\, p(s'|s, a, 0)$ such that $\sum_{s' \in \mathbf{S}} y_s$ is minimal.

Subject to:

(1) $y_s = 1$ *if* $s \in \mathcal{D}$, *and*

(2) $y_s = 0$ *if* $\mathcal{D}$ *is not reachable from* s.

Proof. Given x_s is the maximal probability that an adversarial flow originating at state $s \in \mathbf{S}$ reach a state in $\mathcal{D}$. Then x_s satisfies the constraints (1) and (2). By the definition of x_s, $x_s \geqslant \sum_{s' \in \mathbf{S}} p_D(s', 0)\, , p(s'|s, a, 0) x_{s'}$. This implies that x_s is a feasible solution for the LP. Now we prove that $(x_s)_{s \in \mathbf{S}}$ is the unique solution to the LP.

We prove uniqueness using a contradiction argument. Suppose that $(x_s)_{s \in \mathbf{S}}$ is not the unique solution to the LP. Then there exists $(y_s)_{s \in \mathbf{S}}$ which is a solution to the LP. Since the values of

y_s is minimal under all vectors that satisfy the LP, we get $\sum_{s\in\mathbf{S}} x_s \geqslant \sum_{s\in\mathbf{S}} y_s$. Now we perform value iteration on Problem 6.3.4 by initializing $x_s^{(0)} = y_s$ for all $s \in \mathbf{S}$. Recall that $x_s = \max\{ \sum_{s'\in\mathbf{S}} p_D(s',0)\, p(s'|s,a,0)\, y_{s'} | a \in \mathcal{A}_A(s)\}$ and $y_s \geqslant \sum_{s'\in\mathbf{S}} p_D(s',0)\, p(s'|s,a,0)\, y_{s'}$. This implies that the value iteration yields a decreasing sequence $(x_s^{(n)})_{n\geqslant 0}$. Define $y'_s := \lim_{\to\infty} x_s^{(n)}$. Then $y'_s \leqslant y_s$ and this contradicts the minimality of $\sum_{s\in\mathbf{S}} y_s$. Hence $y'_s = y_s$ and this proves that solution to the LP is unique.

As solution to LP is unique and x_s is a solution, we get $x_s = y_s$ for all $s \in \mathbf{S}$ and this completes the proof. □

Now we give the complexity result of best response calculation of the adversary.

Theorem 6.3.7. The best response of the adversary can be computed in $O(N)$ complexity.

Proof. The linear optimization problem given in Theorem 6.3.6 gives a unique solution which in turn gives best response of the adversary. The LP can be solved in linear time in the number of states using simplex method or any other linear-time LP solver [25], i.e., $O(|\mathbf{S}|)$. As $|\mathbf{S}| = 2 + N$, the complexity of best response calculation of adversary equals $O(N)$. □

6.3.2 Best Response of the Defense

Let the defense strategy be denoted by $p_D = [p_D(s_1), \ldots, p_D(s_N), p_D(\tau), p_D(\phi)]^\intercal$, where $p_D(s) = [p_D(s,0), p_D(s,1)]^\intercal$. The defense best response is an optimal policy $p_D^\star$ associated with the undiscounted infinite-horizon MDP. $p_D^\star$ is a policy that maximizes the long-run average performance. We now give the following definition.

Definition 6.3.8. For a given adversary policy p_A, define $\mathbf{P}_{p_A}(p_D) = [p(s'|s,p_D)]_{s,s'=s_1}^{\phi}$, where

$$p(s'|s,p_D) = \sum_{s'\in\mathbf{S}} \sum_{d\in\{0,1\}} p_A(s'|s,d)\, p_D(s,d).$$

Here $p_A(s'|s,d)$ is the probability of transitioning from s to s' under given policy p_A when the action of the defense is d. Let $\mathbf{r}_D(p_D)$ be the cost vector[1] for the defense under policy p_D.

[1]Then $\mathbf{r}_D(p_D) = [r_D(s_1,p_D), \ldots, r_D(s_N,p_D), r_D(\tau,p_D), r_D(\phi,p_D)]^\intercal$, where $r_D(s,p_D) = \mathcal{C}_D(s) p_D(s,1)$ and $r_D(\tau,p_D) = r_D(\phi,p_D) = 0$.

The expected payoff of the defense at state s and time t for a given adversary strategy p_A is $[\mathbf{P}^t_{p_A}(p_D)r_D(p_D)]_s$. Then the *limiting average value* of the policy p_D is denoted as $\mathbf{v}_D(p_D) := [v_D(s_1, p_D), \ldots, v_D(s_N, p_D), v_D(\tau, p_D), v_D(\phi, p_D)]^\intercal$ is

$$\mathbf{v}_D(p_D) = \lim_{T\to\infty} \Big[\frac{1}{T+1}\sum_{t=0}^{T} \mathbf{P}^t_{p_A}(p_D)\, \mathbf{r}_D(p_D)\Big].$$

Therefore, the best response of the defense is given by the following optimization problem:

Problem 6.3.9. $\max \mathbf{v}_D(p_D)$
Subject to: $p_D \in \mathbf{p}_D$.

We now show that Problem 6.3.9 is equivalent to a linear program (LP) and solving the dual problem of the LP gives a deterministic optimal policy for the defense. Before formulating the LP, we give the following definition.

Definition 6.3.10. For every state $s \in \mathbf{S}$, W_s is an $(N+2)\times 2$ matrix such that the $(s', 1)^{\text{th}}$ entry is $w_{s',1} := 1 - p_A(s'|s, 0)$ and the $(s', 2)^{\text{th}}$ entry is $w_{s',2} := 1 - p_A(s'|s, 1)$ for all $s' \in \mathbf{S}$. Additionally, define vectors

$$\begin{aligned}
\mathbf{x}^\intercal &= [\mathbf{x}^\intercal_{s_1}, \mathbf{x}^\intercal_{s_2}, \ldots, \mathbf{x}^\intercal_{s_N}, \mathbf{x}^\intercal_\tau, \mathbf{x}^\intercal_\phi] \\
\mathbf{y}^\intercal &= [\mathbf{y}^\intercal_{s_1}, \mathbf{y}^\intercal_{s_2}, \ldots, \mathbf{y}^\intercal_{s_N}, \mathbf{y}^\intercal_\tau, \mathbf{y}^\intercal_\phi] \\
\mathbf{r}^\intercal &= [\mathbf{r}^\intercal_{s_1}, \mathbf{r}^\intercal_{s_2}, \ldots, \mathbf{r}^\intercal_{s_N}, \mathbf{r}^\intercal_\tau, \mathbf{r}^\intercal_\phi] \\
\mathbf{J}^\intercal_1 &= [\mathbf{1}^\intercal_{s_1}, \mathbf{0}^\intercal_{s_2} \ldots, \mathbf{0}^\intercal_{s_N}, \mathbf{0}^\intercal_\tau, \mathbf{0}^\intercal_\phi] \\
\mathbf{J}^\intercal_2 &= [\mathbf{0}^\intercal_{s_1}, \mathbf{1}^\intercal_{s_2}, \ldots, \mathbf{0}^\intercal_{s_N}, \mathbf{0}^\intercal_\tau, \mathbf{0}^\intercal_\phi] \\
&\vdots \\
\mathbf{J}^\intercal_\phi &= [\mathbf{0}^\intercal_{s_1}, \mathbf{0}^\intercal_{s_2}, \ldots, \mathbf{0}^\intercal_{s_N}, \mathbf{0}^\intercal_\tau, \mathbf{1}^\intercal_\phi],
\end{aligned}$$

where $\mathbf{x}_s = [x_s(0), x_s(1)]^\intercal$, $\mathbf{y}_s = [y_s(0), y_s(1)]^\intercal$, $\mathbf{r}_s = [0, \mathcal{C}_D(s)]^\intercal$, $\mathbf{1}_s = [1, 1]^\intercal$ and $\mathbf{0}_s = [0, 0]^\intercal$. Further, define $(N+2)\times(2N+4)$ matrices $W := [W_{s_1}\ W_{s_2} \ldots W_{s_N}\ W_\tau\ W_\phi]$, $J := [\mathbf{J}_1\ \mathbf{J}_2 \ldots \mathbf{J}_\tau\ \mathbf{J}_\phi]^\intercal$ and $(N+2)\times 1$ vectors $\mathbf{v} = [v(s_1), \ldots, v(s_N), v(\tau), v(\phi)]^\intercal$ and
$\mathbf{u} = [u(s_1), \ldots, u(s_N), u(\tau), u(\phi)]^\intercal$.

The following Lemma formulates the linear program associated with the Problem 6.3.9.

Lemma 6.3.11. Consider matrices W, J and vectors $\mathbf{u}, \mathbf{v}, \mathbf{x}, \mathbf{y}$, and $\mathbf{r}$ defined in Definition 6.3.10. Then Problem 6.3.9 is equivalent to the following LP formulation: $\min \dfrac{1}{N+2}\mathbf{v}$

Subject to: $\quad [\mathbf{u}^\intercal \ \mathbf{v}^\intercal] \begin{bmatrix} W & 0 \\ J & W \end{bmatrix} \geqslant [\mathbf{r}^\intercal \ \mathbf{0}^\intercal].$

By Proposition 2.9.1 of [50], the proof of Lemma 6.3.11 follows. The dual problem associated with the LP in Lemma 6.3.11 is given below.

Problem 6.3.12. $\max \ \mathbf{r}^\intercal \mathbf{x}$

Subject to: $\quad \begin{bmatrix} W & 0 \\ J & W \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \frac{1}{N+2}\mathbf{1} \end{bmatrix}, \mathbf{x}, \mathbf{y} \geqslant 0.$

Now we present an algorithm to obtain a best response of the defense using Problem 6.3.12.

Algorithm 6 Pseudo-code for best response of defense

1: **Input:** Adversary policy p_A, $W, J, \mathbf{x}, \mathbf{y}$, and $\mathbf{r}$

2: **Output:** Optimal defense policy $p_D^\star$

3: Find an optimal solution $[\mathbf{x}^\star \ \mathbf{y}^\star]^\intercal$ for the dual LP given in Problem 6.3.12

4: Define states set $S^\star := \left\{ s \in \mathbf{S} | x_s^\star = \sum_{i=1}^{2} x_s^\star(i) > 0 \right\}$

5: **if** $s \in S^\star$ **then** select any $i : x_s^\star(i) > 0$

6: **else** $s \notin S^\star$ select any $i : y_s^\star(i) > 0$

7: Define $p_D^\star \in \mathbf{p}_D$ such that $p_D^\star(s, 0) = \begin{cases} 0 \text{ if } i = 1 \\ 1 \text{ if } i = 2 \end{cases}$

8: **end if**

The following theorem shows the output of the Algorithm 6 returns a deterministic best response of the defense for a given adversary policy p_A.

Theorem 6.3.13. Consider Algorithm 6 which takes as input the dual linear program given in Problem 6.3.12 corresponding to a given adversary policy p_A and returns a defense policy $p_D^\star$. Then $p_D^\star$ is an optimal solution to Problem 6.3.9, i.e., a best response for the defense, for the adversary policy p_A.

Proof of Theorem 6.3.13 uses similar arguments of proof of Theorem 2.9.4 in [50] and is omitted here in the interest of space.

6.3.3 Nash Equilibrium of the Game

Existence of Nash equilibrium is not guaranteed for nonzero-sum infinite-horizon stochastic games [50]. Moreover, it is generally difficult to formulate a polynomial-time algorithm to find Nash equilibrium for nonzero-sum infinite-horizon stochastic game. Therefore, in this section we propose an approximate solution method based on nonlinear programming to calculate ϵ-Nash equilibrium strategies for both players $\mathcal{P}_A$ and $\mathcal{P}_D$.

Let U_A^t and U_D^t be the payoffs at time t for $\mathcal{P}_A$ and $\mathcal{P}_D$, respectively. Then the expected payoffs to the players $\mathcal{P}_A$ and $\mathcal{P}_D$ under stochastic behavioral policies (p_A, p_D) at time t and initial state $s_0 \in \mathbf{S}$ is denoted by $\underset{s_0,p_A,p_D}{\mathbb{E}}(U_A^t)$ and $\underset{s_0,p_A,p_D}{\mathbb{E}}(U_D^t)$, respectively. Define discount factor γ_A for $\mathcal{P}_A$ and γ_D for $\mathcal{P}_D$, where $\gamma_A, \gamma_D \in (0,1)$.

We approximate the game introduced in the Section 6.2 to its discounted stochastic game variant as follows. Let the total expected utilities of players $\mathcal{P}_A$ and $\mathcal{P}_D$ for the discounted infinite-horizon game be

$$v_A(s_0, p_A, p_D) = \sum_{t=0}^{\infty} (\gamma_A)^t \left(\underset{s_0,p_A,p_D}{\mathbb{E}}(U_A^t) \right), \text{ and} \tag{6.6}$$

$$v_D(s_0, p_A, p_D) = \sum_{t=0}^{\infty} (\gamma_D)^t \left(\underset{s_0,p_A,p_D}{\mathbb{E}}(U_D^t) \right). \tag{6.7}$$

The players in the discounted stochastic game strategize to maximize their expected payoffs given in Eqs. (6.6) and (6.7). Notice that smaller γ_A, γ_D values implies players are more interested in contemporary payoffs while larger values means players care more about long-term payoffs.

Proposition 6.3.14 (Theorem 3.8.1, Chapter 3, [50]). Every nonzero-sum discounted stochastic game possesses at least one equilibrium point in stationary strategies.

Using Proposition 6.3.14 we present a nonlinear program in Problem 6.3.16 to find an ϵ-Nash equilibrium stochastic stationary strategies $(p_A^\epsilon, p_D^\epsilon)$ for the players $\mathcal{P}_A$ and $\mathcal{P}_D$ of the discounted stochastic game (see Definition 6.3.3).

We first introduce value vectors in Definition 6.3.15 for the players and then explain the notion of ϵ-equilibrium stochastic stationary strategies in Definition 6.3.3. Without loss of generality, we let $\gamma_A = \gamma_D = \gamma$.

Definition 6.3.15. Let the value vector of the adversary $\mathcal{P}_A$ be denoted as

$$\mathbf{v}_A = [v_A(s_1), \ldots, v_A(s_N), v_A(\tau), v_A(\phi)]^\intercal,$$

where each entry represent the expected utility for the adversary at each state $s \in \mathbf{S}$. Similarly, let $\mathbf{v}_D = [v_D(s_1), \ldots, v_D(s_N), v_D(\tau), v_D(\phi)]^\intercal$ be the value vector for the defense $\mathcal{P}_D$. Moreover, values for the both players at the states $s \in \{\mathcal{D}, \tau, \phi\}$ are as follows:

$$(v_A(s), v_D(s)) = \begin{cases} (\beta_A, \beta_D) & \text{if } s \in \mathcal{D} \\ (\alpha_A, \alpha_D) & \text{if } s \in \tau \\ (\sigma_A, \sigma_D) & \text{if } s = \phi \end{cases}$$

Let $m_A(s) = |p_A(s)|$ and $m_D(s) = |p_D(s)|$ denote the number of actions allowed for the adversary and defense at a state $s \in \mathbf{S}$, respectively. Then $m_A = \sum_{s \in \mathbf{S}} |p_A(s)|$ and $m_D = \sum_{s \in \mathbf{S}} |p_D(s)|$ represent the length[2] of stochastic strategies $p_A = [p_A(s_1), \ldots, p_A(s_N), p_A(\tau), p_A(\phi)]$ and $p_D = [p_D(s_1), \ldots, p_D(s_N), p_D(\tau), p_D(\phi)]$, where $p_A(s) = [p_A(s, a_1), \ldots, p_A(s, a_{m_A(s)})]$ and $p_D(s) = [p_D(s, 0), p_D(s, 1)]$ for each $s \in \mathbf{S}$. Note that, $p_D(s, 0), = 1 - p_D(s, 1)$. Now we present the nonlinear programming formulation of the discounted version of the stochastic game.

---

[2]Notice that $m_D(s) = 2$ for all $s \in \mathbf{S}$ and $m_D = 2(N + 2)$ for the player $\mathcal{P}_D$ considered in our game model presented in the Section 6.2.

Problem 6.3.16. Consider the following nonlinear program with the optimization variable $\mathbf{z} = (\mathbf{v}_A, \mathbf{v}_D, p_A, p_D)$:

$$\min \left\{ \sum_{k\in\{A,D\}} \mathbf{1}^T \left[\mathbf{v}_k - \mathbf{r}_k(p_A, p_D) - \gamma \mathbf{P}(p_A, p_D)\mathbf{v}_k\right] \right\}$$

Subject to:

1. $R_A(s)p_D^T(s) + \gamma L(s, \mathbf{v}_A)p_D^T(s) \leqslant v_A(s)\mathbf{1}_{m_A}(s),\ s \in \mathbf{S}$

2. $p_A(s)R_D(s) + \gamma p_A(s)L(s, \mathbf{v}_D) \leqslant v_D(s)\mathbf{1}_{m_D}^T(s),\ s \in \mathbf{S}$

3. $(p_A, p_D) \in \mathbf{p}_A \times \mathbf{p}_D$,

where $R_k(s) = [r_k(s, a, d)]_{a=a_1, d=d_1}^{a_{m_A}(s), d_{m_D}(s)}$ with each entry[3] $r_k(s, a, d)$ gives the cost incurred by the player $\mathcal{P}_k$ with $k \in \{A, D\}$ when $\mathcal{P}_A$ selects action a and $\mathcal{P}_D$ choose action d at the state s. For a given state s and a value vector $\mathbf{v}_k$ for a player (adversary or defense. i.e. $k \in \{A, D\}$),

$$L(s, \mathbf{v}_k) = \left[\sum_{s'\in\mathbf{S}} p(s'|s, a, d)v_k(s')\right]_{a=a_1, d=d_1}^{a_{m_A}(s), d_{m_D}(s)}$$

Furthermore, for $k \in \{A, D\}$ let

$$\mathbf{r}_k(p_A, p_D) = [r_k(s_1, p_A, p_D), \ldots, r_k(\phi, p_A, p_D)]^T$$

with $r_k(s, p_A, p_D) = p_A(s)R_k(s)p_D(s)$ and $\mathbf{P}(p_A, p_D) = [p(s'|s, p_A, p_D)]_{s,s'=s_1}^{\phi}$ with

$$p(s'|s, p_A, p_D) = \sum_{a=a_1}^{a_{m_A}(s)} \sum_{d\in\{0,1\}} p(s'|s, a, d)p_A(s, a)p_D(s, d).$$

Using the existence result from Proposition 6.3.14, we now present the following results on the solution and the convergence of the nonlinear program introduced in the Problem 6.3.16.

---

[3]For the game in Section 6.2, $r_k(s, a, d) = \mathcal{C}_D(s) = c(s)p_D(s, d)$ when $k = \{D\}$ and $d = \{1\}$. For all the other cases $r_k(s, a, d) = 0$. $c(s) < 0$ represents the cost for defense at state s.

Theorem 6.3.17. Consider a point $\hat{\mathbf{z}} = (\hat{\mathbf{v}}_A, \hat{\mathbf{v}}_D, \hat{p}_A, \hat{p}_D)$. Then $(\hat{p}_A, \hat{p}_D)$ in $\hat{\mathbf{z}}$ forms a Nash equilibrium point of the discounted nonzero-sum stochastic game if and only if $\hat{\mathbf{z}}$ is the global minimum of the nonlinear program defined in the Problem 6.3.16 with $\psi(\hat{\mathbf{z}}) = 0$. Let $\psi(\hat{\mathbf{z}})$ be the objective function of nonlinear program for a $\hat{\mathbf{z}}$ satisfying the conditions 1) to 3).

Corollary 6.3.18. Let $\hat{\mathbf{z}}$ be a solution for the nonlinear program defined in the Problem 6.3.16 with an objective function $\psi(\hat{\mathbf{z}}) = \delta > 0$. Then $(\hat{p}_A, \hat{p}_D)$ forms an ϵ-Nash equilibrium with $\epsilon \leqslant \frac{\delta}{1-\gamma}$.

The proof of Theorem 6.3.17 and Corollary 6.3.18 follows from the Theorem 3.8.2 and Corollary 3.8.3, respectively, from Chapter 3 of [50].

6.4 Simulation Study

We perform experimental analysis of our model using NetRecon attack data obtained from RAIN [64] recording system and augmented with conditional branches. The game model in Section 6.2 and the nonlinear algorithm given in Section 9.6 to obtain an ϵ-Nash equilibrium are implemented. The NetRecon attack is designed to perform network reconnaissance on a victim system. The attack starts by executing the *NetRecon* process. Then it gathers information related to network interfaces and IP addresses used by the system and exfiltrates the collected information into a file named */tmp/netrecon.log*.

We first build the *coarse-grained* whole system information flow graph from the attack dataset recorded through the RAIN [64]. Then we identified two nodes in the graph related to NetRecon process, four nodes related to network flow objects (Network interfaces and IP addressee), and the node associated with the file */tmp/netrecon.log* as destinations ($\mathcal{D}$) for the NetRecon attack. Then we prune the coarse-grained graph by starting from the destinations of the attack and building the subgraph related to all the nodes in the coarse-grained graph that have at least one directed path from itself to at least one destination of the attack. We perform our analysis on the resulting refined information flow graph consisting of 22 nodes related to the attack. In order to represent the conditional branches in the system we augmented three selected nodes $s \in V_G \setminus \mathcal{D}$ in the refined graph with pre-defined transition probabilities such that $p(s'|s, a, d) \neq 1$ and $\sum_{s' \in \mathcal{N}(s)} p(s'|s, a, d) =$

1 for each fixed action combinations a and d of the players.

An ϵ-Nash equilibrium of the stochastic game corresponds to a global minimum of Problem 6.3.16 and the function value $\psi(\mathbf{z})$ evaluates to zero (Theorem 6.3.17). The simulations return a local minimum of Problem 6.3.16 at which the function value evaluates to 0.689. Figure 6.1 shows the convergence plot of the simulations.

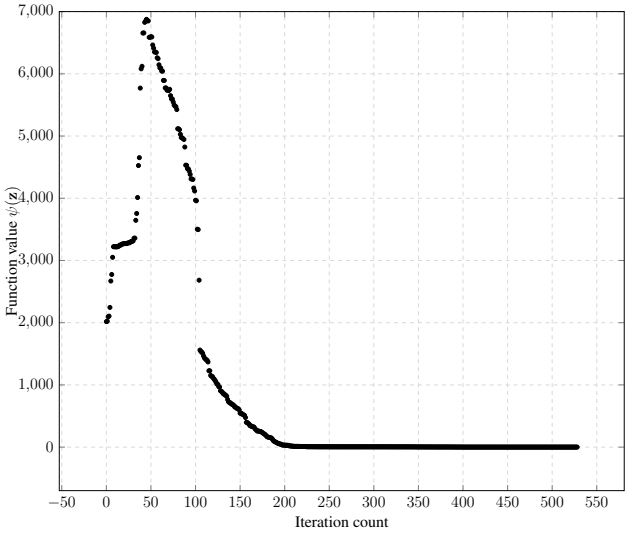
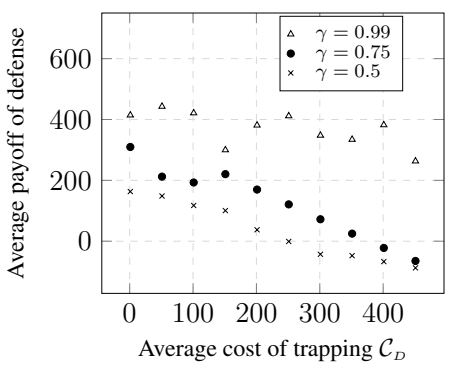
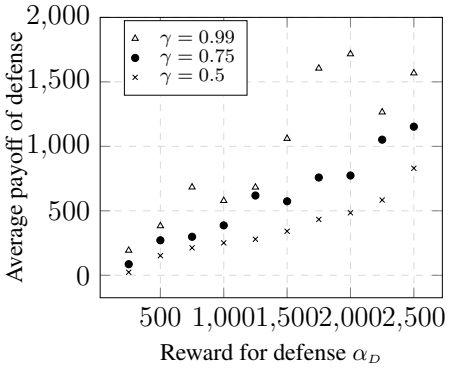


Figure 6.1: Experimental results for NetRecon data obtained using RAIN. Plot shows convergence of the nonlinear program to obtain an ϵ-Nash equilibrium of the game. The cost and reward parameters chosen for the experiment are as follows: $\beta_A = \alpha_D = 500$, $\alpha_A = \beta_D = \sigma_A = -500$, $\sigma_D = 250$ and $\mathcal{C}_D$ values for all the nodes are randomly generated. The discount factor γ is set to 0.99 for both $\mathcal{P}_A$ and $\mathcal{P}_D$. The function value evaluates to 0.689 at the obtained local optimum for the ϵ-Nash equilibrium (at the global optimum ϵ-Nash equilibrium the function value $\psi(\mathbf{z}) = 0$).

In Figure 6.2a we vary the average cost to trap ($\sum_{s \in \mathbf{S}_{\text{eff}}} C_D(s)$, where $\mathbf{S}_{\text{eff}} := \{s \in \mathbf{S} : s \notin \{\phi, \tau\} \cup \mathcal{D}\}$) at the non-terminal states while keeping the other parameters $\beta_A, \beta_D, \alpha_A, \alpha_D, \sigma_A$, and σ_D fixed. Then we plot average cost vs. average payoff of the defense $\sum_{s \in \mathbf{S}_{\text{eff}}} v_D(s)$ at a local optimum returned by the nonlinear program (Problem 6.3.16) for different values of discount factor for both players. With high cost of trapping, the defense decides not to trap at many locations. This increases the probability of winning for the adversary and hence the payoff of the defense



(a) Plot shows variation of average payoff of defense with respect to $\mathcal{C}_D$ and fixed values $\beta_A = \alpha_D = 500$, $\alpha_A = \beta_D = \sigma_A = -500$, and $\sigma_D = 250$.



(b) Plot shows average payoff of defense as a function of α_D with fixed values $\mathcal{C}_D(s) = -100, \forall\, s \in \mathbf{S}_{eff}$, $\beta_A = 500, \beta_D = -\alpha_D$, $\sigma_D = \frac{\alpha_D}{2}$, and $\alpha_A = -500$.

Figure 6.2: Plots showing variation of average payoffs of the defense with respect to variation in the parameters in the payoff functions. All data points in the plots correspond to a local optimum for the nonlinear optimization problem to find ϵ-Nash equilibrium, Problem 6.3.16, for the NetRecon data. Simulations correspond to three different values of the discount factor γ.

decreases. When γ decreases the average payoff to the defense diminishes as γ takes lower values, both players' strategies will be chosen based on the rewards of the immediate states.

For the simulations shown in Figure 6.2b, we first set $\beta_D = -\alpha_D$ and $\sigma_D = \frac{\alpha_D}{2}$. Then we vary defense's reward α_D and plot average payoff of the defense $\sum_{s \in \mathbf{S}_{\text{eff}}} v_D(s)$ for different values of γ. We keep the rest of the parameters $\mathcal{C}_D, \beta_A, \alpha_A$, and σ_A fixed. As α_D increases, the average payoff of the defense increases as expected. The simulations also suggests that lower discount factors result in lower average payoff for the defense.

6.5 Summary

In this chapter, we modeled Dynamic Information Flow Tracking (DIFT) to detect Advanced Persistent Threats (APTs) in a system. The key focus of this chapter is to obtain an analytical model for DIFT that can track information flows arising from data-flow commands and also conditional

branches in the system. We modeled the interaction between APTs and DIFT as a stochastic nonzero-sum game in infinite-horizon and stochastic strategies. We performed best response analysis for both players. The best response of the adversary is a *maximal reachability probability problem* formulated and solved as a linear program in this chapter. The best response of the defense is formulated as a *linear program* and calculated using a polynomial-time algorithm. Existence of Nash equilibrium for nonzero-sum infinite-horizon stochastic games is unknown and if exists its computation is difficult. Hence we considered a discounted version of the problem. It is guaranteed that there exists a Nash equilibrium for a nonzero-sum discounted stochastic game. We proposed a *nonlinear programming* problem to calculate ϵ-Nash equilibrium using a value iteration algorithm. Finally, we tested the proposed model and algorithms on augmented (conditional branches) version of real-world data for NetRecon attack obtained using Refinable Attack INvestigation (RAIN) [64] framework. Investigating existence of Nash equilibrium for our undiscounted game model and analyzing the game when the states are unobservable by the defender are part of future work.

Chapter 7

DISCOUNTED STOCHASTIC DIFT-APT GAMES WITH REINFORCEMENT LEARNING

7.1 Motivation

In a system equipped with DIFT, interactions of the APT with the system to achieve the malicious objective while evading detection depends on the efficiency of the DIFT scheme. On the other hand, determining a resource-efficient policy for DIFT that maximize the detection probability depends on the nature of APT's interactions with the system. Non-cooperative game theory provides rich set of rules that can model the strategic interactions between two competing agents (APT and DIFT). The feasible interactions among system processes and objects (e.g., files, network sockets) can be abstracted into a graph called *information flow graph* [64]. Therefore, in this chapter we propose a game-theoretic formulation on the underlying system information flow graph to facilitate the study of trade-off between resource efficiency and efficiency of detection in DIFT. We make the following contributions in this chapter.

- We model the interaction between APT and DIFT as a two-player, nonzero-sum, imperfect information, infinite-horizon *stochastic game*. The state space of the game and the action sets of both players are finite and the players strategize from a stationary policy to maximize their individual payoffs.

- We capture the performance evaluation metrics such as false negatives of the DIFT using the stochastic nature of the game. Our attack model allows APT to continuously attack the system while having the capability of relaunching the attack in case it fails to achieve its final objective.

- We provide a reinforcement learning based algorithm that converges to a Nash equilibrium

(NE) of the APT vs. DIFT stochastic game. The proposed algorithm utilizes the structure of the game and is based on the two-time scale algorithm in [104].

- To evaluate the performance of our approach, we perform experimental analysis of the model and the algorithm on nation state attack data obtained from Refinable Attack INvestigation (RAIN) [64] framework.

7.2 Stochastic DIFT-APT Game Formulation

In this section, we model a two-player stochastic game between APTs ($\mathcal{P}_A$) and DIFT ($\mathcal{P}_D$). Let N be the number of nodes in the IFG, $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$, M be the number of stages of the attack, and $\lambda \subset V_\mathcal{G}$ denote the set of possible entry points for the APT. We introduce a virtual-node s_0 and a set of out-going edges E_0 from s_0 to each node in the set λ to model the entry point of the APT in the system. The resulting IFG is referred to as Modified Information Flow Graph (MIFG), $\hat{\mathcal{G}} = (V_{\hat{\mathcal{G}}}, E_{\hat{\mathcal{G}}})$, where $V_{\hat{\mathcal{G}}} = V_\mathcal{G} \cup s_0$ and $E_{\hat{\mathcal{G}}} = E_\mathcal{G} \cup E_0$.

7.2.1 System and Player Models

Consider a system equipped with a DIFT defense scheme ($\mathcal{P}_D$) threatened by an APT ($\mathcal{P}_A$). Recall, $\mathcal{D}_j \subset V_\mathcal{G}$, for $j = 1, \ldots, M$, denote the nodes in the IFG corresponding to the goals of the j^{th} attack stage. We refer to the set $\mathcal{D}_j$ as the destination nodes of the APT ($\mathcal{P}_A$) in the stage j. The objective of $\mathcal{P}_A$ is to sequentially reach at least one node from each set $\mathcal{D}_j$, for all $j = 1, \ldots, M$, without getting detected by the DIFT ($\mathcal{P}_D$). We model the multi-stage attack of the APT as a multi-stage dynamic game between $\mathcal{P}_D$ and $\mathcal{P}_A$.

The information flows originating from the set λ are tainted as suspicious flows by the DIFT. In order to detect the presence of the APTs and mitigate the threats imposed by APTs, DIFT inspects the tainted flows at certain processes and objects in the system (i.e., $V_\mathcal{G}$). The cost of performing security analysis at a node of the IFG varies depending on the available resources (e.g., memory, processing power) and the intensity of the traffic (amount of benign and malicious information flows) at each node of the IFG. Hence, the objective of $\mathcal{P}_D$ is to identify a set of nodes in the IFG to

perform security analysis to prevent $\mathcal{P}_A$ from successfully completing the attack while minimizing the resource cost.

The game between the two players $\mathcal{P}_A$ and $\mathcal{P}_D$ unfolds in time $t \in \mathcal{T} := \{1, 2, \ldots\}$ on the MIFG with player $\mathcal{P}_A$ starting the game at time $t = 1$ from node s_0 in the MIFG. As APTs are persistent attackers, we consider an infinite horizon game where $\mathcal{P}_A$ is allowed to restart the attack whenever one of the following conditions are satisfied:

1. Player $\mathcal{P}_A$ sequentially reach at least one node in set $\mathcal{D}_j$, for all $j = 1, \ldots, M$.

2. $\mathcal{P}_A$ drops out of the game (abandoning the attack).

3. Player $\mathcal{P}_D$ successfully detect $\mathcal{P}_A$ before $\mathcal{P}_A$'s objective is achieved.

In case 1 the adversary chooses to continue the successful attack, case 2 the attacker abandons the current attack and starts a new attack, and case 3 the attacker starts a new attack as the defender detected the attacker in the current attack.

7.2.2 State Space of the Game

Let $\mathbf{S} := \{s_0\} \cup \{V_\mathcal{G} \times \{1, \ldots, j\}\} \cup \{\{\phi, \tau\} \times \{1, \ldots, j\}\} = \{s_0, s_1^j, \ldots, s_N^j, \phi^j, \ldots, \tau^j\}$, for all $j \in \{1, \ldots, M\}$, represent the finite state space of the multi-stage game. $s_1, \ldots, s_N$ indicate the nodes of the IFG, j denotes the stage of the game, and s_i^j denotes a state where tagged flow is at node s_i in stage j. States ϕ^j and τ^j are corresponding to $\mathcal{P}_A$ dropping out of the game and $\mathcal{P}_D$ successfully detecting $\mathcal{P}_A$, respectively, at stage j. Additionally, the states s_i^M with $s_i \in \mathcal{D}_M \subset V_\mathcal{G}$, where $i = 1, \ldots, N$, are associated with $\mathcal{P}_A$ achieving the final goal. Let the random variable $\bar{s}_t$ denote the *state* of the game at time $t \in \mathcal{T} := \{1, 2, \ldots\}$.

Let $\mathcal{A}_A$ and $\mathcal{A}_D$ denote the action spaces of players $\mathcal{P}_A$ and $\mathcal{P}_D$, respectively. At each time $t \in \mathcal{T}$, $\mathcal{P}_A$ and $\mathcal{P}_D$ *simultaneously* take the actions $a_t \in \mathcal{A}_A$ and $d_t \in \mathcal{A}_D$, respectively. The action sets of players $\mathcal{P}_A$ and $\mathcal{P}_D$ at any state $s \in \mathbf{S}$, $\mathcal{A}_A(s)$ and $\mathcal{A}_D(s)$, can be defined as five cases. Case (i): when $s = s_i^1$, where $s_i \in \lambda$. Then, $\mathcal{A}_A(s) = \{s_{i'}^1 : (s_i, s_{i'}) \in E_\mathcal{G}\} \cup \{\varnothing\}$, adversary can

select an action to move to a node $s_{i'}$ in stage one and $\mathcal{A}_D(s) = 0$, defender is not allowed to trap tagged flows. Case (ii): when $s = s_i^j$, where $s_i \notin \mathcal{D}_j$. Then, $\mathcal{A}_A(s) = \{s_{i'}^1 : (s_i, s_{i'}) \in E_{\mathcal{G}}\} \cup \{\varnothing\}$ which illustrates player $\mathcal{P}_A$ choosing to transition to one of the out-neighboring node of s_i in stage j and action $\varnothing$ represents $\mathcal{P}_A$ dropping out of the game. Also, $\mathcal{A}_D(s) \in \{0, 1\}$, where the action $d_t =$ "0" and $d_t =$ "1" denote $\mathcal{P}_D$ deciding not to trap tagged flows and deciding to trap tagged flows, respectively. Case (iii): when $s = s_i^j$, where $s_i \in \mathcal{D}_j$ for $j = 1, \ldots, M-1$. Then, $\mathcal{A}_A(s) = \{s_i^{j+1}\}$ which represents APT traversing from stage j of the attack to stage $j+1$ and $\mathcal{A}_D(s) = 0$. Case (iv): when $s = s_0$, $\mathcal{A}_A(s) = \{s_{i'}^1 : s_{i'} \in \lambda\}$ and $\mathcal{A}_D(s) = \{0\}$. Case (v): when $s = \{\phi, \tau\} \cup \{s_i^M : s_i \in \mathcal{D}_M\}$. Then, $\mathcal{A}_A(s) = s_0$ which captures the ability of APT to comeback and relaunch the attack in the system and $\mathcal{A}_D(s) = 0$.

We assume state transitions are stationary, i.e., state at time $t+1$, $\bar{s}_{t+1}$ depends only on the current state $\bar{s}_t$ and the actions a_t and d_t of both players at the state $\bar{s}_t$. The stationary transition probability from a given state s to a state s' under actions $a \in \mathcal{A}_A(s)$ and $d \in \mathcal{A}_D(s)$ is defined as follows:

$$p(s'|s, a, d) := \mathbb{P}\{\bar{s}_{t+1} = s'|\bar{s}_t = s, a_t = a, d_t = d\}, \tag{7.1}$$

for all $s, s' \in \mathbf{S}$, where $p : \mathbf{S} \times \mathbf{S} \times \mathcal{A}_A \times \mathcal{A}_D \to [0, 1]$. Note that, for given $\bar{s}_t = s, a_t = a$, and $d_t = d$, the state $\bar{s}_{t+1} = s'$ is assumed to be conditionally independent of all other random variables at a time less than or equal to t. Further, $\sum_{s' \in \mathbf{S}} p(s'|s, a, d) = 1$. Moreover, $p(s'|s, a, d) = 1$ in the following two cases: (i) when $d = 0$ and $a = s' = \mathcal{A}_D(s)$ and (ii) when $d = 1$, $a = \varnothing$ and $s' = \phi$. Consider a state $\bar{s}_t = s$ with defender's action $d_t = 1$ and adversary's action $a_t \neq \varnothing$. Then $0 < p(s'|s, a, d) = p(s'|s, a, 1) < 1$ due to the false negatives of the DIFT[1]. Let $f_n(s_i^j)$ denote the rate of false negatives generated at node s_i of the IFG in stage j. Then for a state $\bar{s}_t = s_i^j$, action $a_t = s_{i'}^{j'}$ and $d_t = 1$ the possible next state $\bar{s}_{t+1}$ are

$$\bar{s}_{t+1} = \begin{cases} s_{i'}^{j'} & \text{with probability } f_n(s_i^j), \\ \tau & \text{with probability } 1 - f_n(s_i^j). \end{cases} \tag{7.2}$$

[1]When $d = 1$, $p(s'|s, a, d)$ is expected to be strictly 1 with $s' = \tau$ for any choice of $a \neq \varnothing$ if DIFT has no false negative rates. In this chapter we assume that DIFT has some nonzero rate of false negatives.

False negatives of the DIFT scheme arise from the limitations of the security rules that can be deployed at each node of the IFG (i.e., processes and objects in the system). Such limitations are due to variations in the number of rules and the depth of the security analysis[2] (e.g., system call level trace, CPU instruction level trace) that can be implemented at each node of the IFG resulting from the resource constraints including memory, storage and processing power imposed by the system on each IFG node.

We assume both players know the current state of the game. But $\mathcal{P}_D$ is unaware whether an incoming tagged (suspicious) flow is malicious or not and $\mathcal{P}_A$ does not know the chances of getting trapped at different processes and objects in the system. This results in an *asymmetry* on the information possessed by each players. Hence the game is an *imperfect* information game. Furthermore, both players are unaware of the transition probabilities which depend on the rate of false negatives generated at the different nodes of the IFG (Eq. (7.2)). Consequently, player's do not know the payoff structure of the game and hence the game is an *incomplete* information game.

7.2.3 Policies and Payoffs of the Players

Players $\mathcal{P}_A$ and $\mathcal{P}_D$ decide their actions based on the current state of the game. Thus, the policies of the players we consider here are restricted to *stationary* policies[3]. At any time $t \in \mathcal{T}$, $\mathcal{P}_A$ and $\mathcal{P}_D$ select an action from the action set $\mathcal{A}_A$ and $\mathcal{A}_D$, respectively, based on some probability distribution. Hence, the player policies are *stochastic* policies[4]. A stochastic stationary policy of player $\mathcal{P}_A$ is given by $p_A : \mathbf{S} \rightarrow [0,1]^{|\mathcal{A}_A|}$ and of player $\mathcal{P}_D$ is given by $p_D : \mathbf{S} \rightarrow [0,1]^{|\mathcal{A}_D|}$. The set of all stationary, stochastic policies of $\mathcal{P}_A$ and $\mathcal{P}_D$ are denoted by $\mathbf{p}_A$ and $\mathbf{p}_D$, respectively. Let $\mathbf{P}(p_A, p_D)$ represent the state transition matrix of the game resulting from $p_A \in \mathbf{p}_A, p_D \in \mathbf{p}_D$. Then,

$$\mathbf{P}(p_A, p_D) = \left[p(s'|s, p_A, p_D)\right]_{s,s' \in \mathbf{S}}, \text{ where}$$

---

[2]Detecting an unauthorized use of tagged flow crucially depends on the path traversed by the information flow.

[3]A policy of a player is called as stationary, if the player's decision of choosing an action in any state $s \in \mathbf{S}$ is invariant with respect to the time of visit to s [50].

[4]In contrast, a policy $p_A \in \mathbf{p}_A$ ($p_D \in \mathbf{p}_D$) is said to be a *pure* or *deterministic* policy if for all $s \in \mathbf{S}$, the entries of vectors p_D and p_A belong to the set $\{0, 1\}$.

$$p(s'|s,p_A,p_D)=\sum_{a\in\mathcal{A}_A(s)}\sum_{d\in\mathcal{A}_D(s)}p(s'|s,a,d)p_A(s,a)p_D(s,d).$$

Next define $\mathbf{r}_A(p_A,p_D)$ ($\mathbf{r}_D(p_A,p_D)$) to be the expected reward vector of $\mathcal{P}_A$ ($\mathcal{P}_D$) under the policies $p_A\in\mathbf{p}_A$ and $p_D\in\mathbf{p}_D$. Then for $k\in\{A,D\}$ let

$$\mathbf{r}_k(p_A,p_D)=\Big[r_k(s,p_A,p_D)\Big]_{s\in\mathbf{S}}=\Big[r_k(s_0,p_A,p_D),\ldots,r_k(\tau^M,p_A,p_D)\Big]^T, \tag{7.3}$$

where $r_k(s,p_A,p_D)=\sum\limits_{\substack{a\in\mathcal{A}_A(s)\\ d\in\mathcal{A}_D(s)}}r_k(s,a,d)p_A(s,a)p_D(s,d)$ for each $s\in\mathbf{S}$. Furthermore, $r_A(s,a,d)$ and $r_D(s,a,d)$ are defined as follows.

$$r_A(s,a,d) = \begin{cases} \alpha_A^j & \text{if } s=\tau^j \\ \beta_A^j & \text{if } s=s_i^j : s_i\in\mathcal{D}_j \\ \sigma_A^j & \text{if } s=\phi^j \\ 0 & \text{otherwise} \end{cases} \tag{7.4}$$

$$r_D(s,a,d) = \begin{cases} \alpha_D^j & \text{if } s=\tau^j \\ \beta_D^j & \text{if } s=s_i^j : s_i\in\mathcal{D}_j \\ \sigma_D^j & \text{if } s=\phi^j \\ \mathcal{C}_D(s) & \text{if } s=s_i^j, s_i\notin\{\mathcal{D}_j\cup\lambda\} \text{ and } d=1 \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

Here $r_k(s,a,d)$ gives the reward associated with player $\mathcal{P}_k$, where $k\in\{A,D\}$, when $\mathcal{P}_A$ selects action a and $\mathcal{P}_D$ choose action d at the state s. Reward for $\mathcal{P}_A$ at a state s when each player chooses their respective actions $a\in\mathcal{A}_A(s)$ and $d\in\mathcal{A}_D(s)$, $r_A(s,a,d)$, consists of three components (i) penalty term $\alpha_A^j<0$ if the APT is detected by the defender in the j^{th} stage where $\alpha_A^1\leq\ldots\leq\alpha_A^M$, (ii) reward term $\beta_A^j>0$ for APT reaching a destination of stage j, for $j=1,\ldots,M$, where $\beta_A^1\leq\ldots\leq\beta_A^M$, and (iii) penalty term $\sigma_A^j<0$ for APT dropping out of the game in the j^{th} stage where $\sigma_A^1\leq\ldots\leq\sigma_A^M$. On the other hand $r_D(s,a,d)$ consists of four components (i) reward term $\alpha_D^j>0$ for defender detecting the APT in the j^{th} stage where $\alpha_D^1\geq\ldots\geq\alpha_D^M$,

(ii) penalty term $\beta_D^j < 0$ for APT reaching a destination of stage j, for $j = 1, \ldots, M$, where $\beta_D^1 \geq \ldots \geq \beta_D^M$, (iii) reward $\sigma_D^j > 0$ for APT dropping out of the game in the j^{th} stage where $\sigma_D^1 \geq \ldots \geq \sigma_D^M$, and (iv) a security cost $\mathcal{C}_D(s) < 0$ associated with performing security checks on tagged flows at a state $s = s_i^j : s_i \notin \{\mathcal{D}_j \cup \lambda\}$, i.e., cost of choosing node $s_i \notin \{\mathcal{D}_j \cup \lambda\}$ in stage $j \in \{1, \ldots, M\}$ as a tag sink.

Remark 7.2.1. Security cost, $\mathcal{C}_D(s)$, at a state, $s = \{s_i^j : s_i \notin \{\mathcal{D}_j \cup \lambda\}\}$, consists of two components:

1. Resource cost for performing security checks on tagged benign flows.
   Let $c_1(j)$ denote the fixed resource cost incurred to the system if security check is done for tagged information flows reaching each node s_i in stage j. Define $p_f(s_i^j)$ to be the fraction of tagged flows through node s_i in stage j. Then the resource cost associated with performing security analysis at a state s can be written as $c_1(j)p_f(s_i^j)$.

2. Cost of false alarms or false positives (cost of identifying a tagged benign flow as a malicious flow).
   Let $f_p(s_i^j)$ be the false positive rate if DIFT performs security analysis at a node s_i in the stage j. Also, let $c_2(j)$ denote the fixed cost associated with generation of false alarms in the system at stage j, i.e., if a tagged benign flows at node s_i during stage j triggers a false alarm. Then the expected cost of false alarm at a state s is given by $f_p(s_i^j)c_2(j)p_f(s_i^j)$.

Hence, $\mathcal{C}_D(s) = c_1(j)p_f(s_i^j) + f_p(s_i^j)c_2(j)p_f(s_i^j)$.

Let the payoffs of players $\mathcal{P}_A$ and $\mathcal{P}_D$ at time t be denoted as $U_A(t)$ and $U_D(t)$, respectively. Then $\underset{s_0,p_A,p_D}{\mathbb{E}}(U_A(t))$ and $\underset{s_0,p_A,p_D}{\mathbb{E}}(U_D(t))$ characterize the expected payoffs to players $\mathcal{P}_A$ and $\mathcal{P}_D$, respectively, at time t. Moreover, for $k \in \{A, D\}$, $\underset{s_0,p_A,p_D}{\mathbb{E}}(U_k(t)) = \left[\mathbf{P}^t(p_A, p_D)\mathbf{r}_k(p_A, p_D)\right]_{s_0}$. Next we define the *discounted value* ($v_k(s_0, p_A, p_D)$), i.e., discounted expected payoff, of the game for players ($k \in \{A, D\}$) under the initial state s_0 and stationary player policies p_A and p_D.

$$v_k(s_0, p_A, p_D) = \sum_{t=0}^{\infty} \gamma^t \left(\mathop{\mathbb{E}}_{s_0, p_A, p_D} (U_k(t)) \right) \tag{7.6}$$

Here $\gamma \in [0, 1)$ denote the discount factor of the game. Notice that smaller (close to '0') γ in Eq. (7.6) implies players are more interested in short-term rewards while larger values (close to '1') mean players are more concerned about long-term rewards. However, APTs are specifically designed to launch long-term stealthy attacks on victim systems. Hence we highlight here that using $\gamma \rightarrow 1$ is essential for approximately capturing the true long-term aspects of the game between APT and DIFT. Please refer to Remark 7.3.1 for more information on the choice of using discounting to define the values of players.

Let Γ denote the game between $\mathcal{P}_A$ and $\mathcal{P}_D$. Note that Γ is a *non-cooperative* game where both players try to maximize their respective payoffs defined in Eq. (7.6). The differences of the components in $r_A(s, a, d)$ and $r_D(s, a, d)$ makes $v_A(s_0, p_A, p_D) \neq -v_D(s_0, p_A, p_D)$ generally. Hence Γ is a nonzero-sum stochastic game. Moreover, due to the discount factor, γ, used in $v_k(s_0, p_A, p_D)$, Γ is a discounted stochastic game.

Finally let $\rho(\Gamma)$ define the set of parameters associated with Γ. Game parameters include $c_1(j)$, $c_2(j)$, $p_f(s_i^j)$, $\mathcal{C}_D(s_i^j)$, $f_n(s_i^j)$, $f_p(s_i^j)$, α_A^j, β_A^j, σ_A^j, α_D^j, σ_D^j and β_D^j for $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, M\}$. Some of the game parameters such as $f_n(s_i^j)$ are difficult to know a priori. In Section 7.4 we discuss how our proposed algorithm successfully tackles such scenarios where players lack knowledge on some parameters in $\rho(\Gamma)$ (specifically the transition probability structure of the game). Moreover, in Section 7.5 we provide details on extracting cost parameters (e.g., $\mathcal{C}_D(s_i^j)$) from system log data and estimating reward/penalty parameters.

7.3 Solution to Stochastic DIFT-APT Game

In this section, we first introduce the notion of equilibrium considered in the chapter. Then we briefly discuss computation of equilibrium policies in infinite-horizon stochastic games.

We use the solution concept of *Nash equilibrium* (NE) to analyze Γ. A stochastic stationary

strategy profile $(p_D^\star, p_A^\star)$ is a NE if

$$v_A(s_0, p_A, p_D^\star) \leq v_A(s_0, p_A^\star, p_D^\star) \text{ for all } p_A \in \mathbf{p}_A, \text{ and} \tag{7.7}$$

$$v_D(s_0, p_A^\star, p_D) \leq v_D(s_0, p_A^\star, p_D^\star) \text{ for all } p_D \in \mathbf{p}_D. \tag{7.8}$$

Remark 7.3.1. Computation of NE in infinite-horizon stochastic games depends on the payoff evaluation criteria used by the players. Discounted reward method and limiting average (also referred as undiscounted[5] reward criteria) are two payoff evaluation procedures widely used in stochastic games. Existence of a Nash equilibrium for a general two-player nonzero-sum, undiscounted, infinite-horizon, general stochastic game under stationary polices remain as an open problem. [133] provides an illustrative example of an undiscounted stochastic game where no stationary strategy exists for both players even when a weaker notion of NE is considered. However, Nash equilibrium of discounted stochastic games under stationary policies is well studied in the literature (see Proposition 7.3.2) and there exists a nonlinear program that can be used to calculate stationary NE policies (see Problem 7.3.3).

The following proposition provides the existence of a Nash equilibrium for two player nonzero-sum discounted stochastic games.

Proposition 7.3.2 ([50], Ch. 3, Theorem 3.8.1). Every nonzero-sum discounted stochastic game has at least one Nash equilibrium point in stationary strategies.

Let the value vector of a player $k \in \{A, D\}$ be denoted by

$$\mathbf{v}_k = \Big[v_k(s)\Big]_{s\in\mathbf{S}} = \begin{bmatrix} v_k(s_0) & \dots & v_k(\tau^M) \end{bmatrix}^T,$$

where each entry represent the expected payoff for the k^{th} player at some state $s \in \mathbf{S}$. We present the following nonlinear program (NLP) adopted from [50] to characterize NE in Γ.

---

[5] *Undiscounted value* of a game for players ($k \in \{A, D\}$) under the initial state s_0 and stationary player policies p_A and p_D is defined by $v_k(s_0, p_A, p_D) = \lim_{T\to\infty}\Big[\frac{1}{T+1}\sum_{t=0}^{T}\Big(\mathop{\mathbb{E}}_{s_0,p_A,p_D}(U_k(t))\Big)\Big]$.

Problem 7.3.3. Consider the following NLP with the optimization variable $\mathbf{z} = (\mathbf{v}_A, \mathbf{v}_D, p_A, p_D)$:

$$\min \left\{ \sum_{k \in \{A,D\}} \mathbf{1}^T \left[\mathbf{v}_k - \mathbf{r}_k(p_A, p_D) - \gamma \mathbf{P}(p_A, p_D) \mathbf{v}_k \right] \right\}$$

Subject to:

1. $(p_A, p_D) \in \mathbf{p}_A \times \mathbf{p}_D$,

2. $R_A(s) p_D^T(s) + \gamma L(s, \mathbf{v}_A) p_D^T(s) \leqslant v_A(s) \mathbf{1}_{m_A}(s)$, $s \in \mathbf{S}$

3. $p_A(s) R_D(s) + \gamma p_A(s) L(s, \mathbf{v}_D) \leqslant v_D(s) \mathbf{1}_{m_D}^T(s)$, $s \in \mathbf{S}$

where $R_k(s) = [r_k(s, a, d)]_{a \in \mathcal{A}_A(s), d \in \mathcal{A}_D(s)}$ with $m_A(s) = |p_A(s)|$ and $m_D(s) = |p_D(s)|$ denoting the number of actions allowed for $\mathcal{P}_A$ and $\mathcal{P}_D$ at a state $s \in \mathbf{S}$, respectively. $\mathbf{1}_i$ is a column vector of all ones of size i. Moreover, for a given state s and a value vector $\mathbf{v}_k$ for a player $k \in \{A, D\}$,

$$L(s, \mathbf{v}_k) = \left[\sum_{s' \in \mathbf{S}} p(s'|s, a, d) v_k(s') \right]_{a \in \mathcal{A}_A(s), d \in \mathcal{A}_D(s)}.$$

In Problem 7.3.3, Condition 1 ensures (p_A, p_D) is a valid stochastic stationary policy pair of Γ that satisfies the basic probability theory conditions such as for all $s \in \mathbf{S}$, $\sum_{a \in \mathcal{A}_A(s)} p_A(s, a) = 1$ with $p_A(s, a) \geq 0$ and $\sum_{d \in \mathcal{A}_D(s)} p_D(s, d) = 1$ with $p_D(s, d) \geq 0$. Whereas Conditions 2 and 3 ensure a valid policy pair (p_A, p_D) is a NE of Γ. Let the objective function of Problem 7.3.3 be denoted by $\psi(\mathbf{z})$. Then Theorem 3.8.2 in [50] states that $\psi(\mathbf{z}) = 0$ at NE. However it is noted that if the underlying non-zero sum game has a local minimum which is not a global minimum then NLP presented in Problem 7.3.3 can get stuck at a local minima which prevents $\psi(\mathbf{z})$ from reaching to 0 [104]. Hence in such cases the solution of Problem 7.3.3 will not converge to an NE.

A Reinforcement Learning (RL) algorithm, named *ON-SGSP*, is proposed in [104] to solve the NLP in Problem 7.3.3 which is guaranteed to converge to a NE when the following assumption is satisfied.

Assumption 7.3.4. In Γ, Markov chain induced by the state transition matrix $\mathbf{P}(p_A, p_D)$ is irreducible and positive recurrent under all possible player policies $p_A \in \mathbf{p}_A$ and $p_D \in \mathbf{p}_D$.

Let the current state be $\bar{s}_t$ and a_t (d_t) be the action chosen by $\mathcal{P}_A$ ($\mathcal{P}_D$). Let $v_A^t(\bar{s}_t)$ ($v_D^t(\bar{s}_t)$), $\zeta_A^t(\bar{s}_t, a_t)$ ($\zeta_D^t(\bar{s}_t, d_t)$) and $p_A^t(\bar{s}_t, a_t)$ ($p_D^t(\bar{s}_t, d_t)$) denote the discounted value, estimated gradient of $\psi(\mathbf{z})$ and probability of action $a_t \in \mathcal{A}_A(\bar{s}_t)$ ($d_t \in \mathcal{A}_D(\bar{s}_t)$) at time t for player $\mathcal{P}_A$ ($\mathcal{P}_D$), respectively. Then the following set of update equations captures the core steps of the ON-SGSP algorithm in [104].

Value updates:

$$v_A^{t+1}(\bar{s}_t) = v_A^t(\bar{s}_t) + c(t)[\hat{\epsilon}_A(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)]$$
$$v_D^{t+1}(\bar{s}_t) = v_D^t(\bar{s}_t) + c(t)[\hat{\epsilon}_D(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)]$$

Gradient estimations:

$$\begin{aligned}\zeta_A^{t+1}(\bar{s}_t, a_t) &= \zeta_A^t(\bar{s}_t, a_t) + c(t)\left[\sum_{k\in\{A,D\}}(\hat{\epsilon}_k(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)) - \zeta_A^t(\bar{s}_t, a_t)\right] \\ \zeta_D^{t+1}(\bar{s}_t, d_t) &= \zeta_D^t(\bar{s}_t, d_t) + c(t)\left[\sum_{k\in\{A,D\}}(\hat{\epsilon}_k(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)) - \zeta_D^t(\bar{s}_t, d_t)\right]\end{aligned} \tag{7.9}$$

Policy updates:

$$p_A^{t+1}(\bar{s}_t, a_t) = \Pi\left(p_A^t(\bar{s}_t, a_t) - b(t)\sqrt{p_A^t(\bar{s}_t, a_t)} \times |\hat{\epsilon}_A(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)|\overline{\text{sgn}}(-\zeta_A^{t+1}(\bar{s}_t, a_t))\right)$$
$$p_D^{t+1}(\bar{s}_t, d_t) = \Pi\left(p_D^t(\bar{s}_t, d_t) - b(t)\sqrt{p_D^t(\bar{s}_t, d_t)} \times |\hat{\epsilon}_D(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)|\overline{\text{sgn}}(-\zeta_D^{t+1}(\bar{s}_t, d_t))\right),$$

where

$$\begin{aligned}\hat{\epsilon}_A(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t) &= r_A(\bar{s}_t, a_t, d_t) + \gamma v_A^t(\bar{s}_{t+1}) - v_A^t(\bar{s}_t) \\ \hat{\epsilon}_D(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t) &= r_D(\bar{s}_t, a_t, d_t) + \gamma v_D^t(\bar{s}_{t+1}) - v_D^t(\bar{s}_t)\end{aligned} \tag{7.10}$$

for all $\bar{s}_t \in \mathbf{S}$ and $t \in \mathcal{T}$. Here, the function $\overline{\text{sgn}}(x)$ denotes the continuous version of the sign/signum function and it maps any x outside of a very small interval around 0 to +1 or -1 depending on the sign of the x [104]. Function $\Pi(.)$ projects the policy $p_A^t(\bar{s}_t)$ ($p_D^t(\bar{s}_t)$) into the simplex defined by $p_A^t(\bar{s}_t, a_t) > 0$ for all $a_t \in \mathcal{A}_A(\bar{s}_t)$ ($p_D^t(\bar{s}_t, d_t) > 0$ for all $d_t \in \mathcal{A}_D(\bar{s}_t)$) and $\sum_{a_t\in\mathcal{A}_A(\bar{s}_t)} p_A^t(\bar{s}_t, a_t) = 1$ ($\sum_{d_t\in\mathcal{A}_D(\bar{s}_t)} p_D^t(\bar{s}_t, d_t) = 1$). The terms $b(t)$ and $c(t)$ denote the two time scale learning rates/step-sizes of the RL algorithm that satisfies $\lim_{t\to\infty}\sup\frac{b(t)}{c(t)} \to 0$. This implies that the

value update and the gradient estimation occur in a faster time scale compared to the policy update. Further, $b(t)$ and $c(t)$ satisfy the standard step-size conditions that are required for the convergence of two time scale algorithms [29] such as $\sum_{t=0}^{\infty} b(t) = \sum_{t=0}^{\infty} c(t) = \infty$ and $\sum_{t=0}^{\infty} b^2(t) = \sum_{t=0}^{\infty} c^2(t) < \infty$.

Construction of update equations in Eq. (7.9) is based on the Bellman error. The following equations define the Bellman error $\epsilon_{A}(s, a, p_{D})$ ($\epsilon_{D}(s, d, p_{A})$) of $\mathcal{P}_{A}$ ($\mathcal{P}_{D}$) at state $s \in \mathbf{S}$ when action $a \in \mathcal{A}_{A}(s)$ ($d \in \mathcal{A}_{D}(s)$) is used while $\mathcal{P}_{D}$ ($\mathcal{P}_{A}$) is following the policy p_{D} (p_{A}).

$$\begin{aligned}
\epsilon_{A}(s, a, p_{D}) &= \sum_{d \in \mathcal{A}_{D}(s)} \left(r_{A}(s, a, d) + \gamma \sum_{s' \in \mathbf{S}} p(s'|s, a, d) v_{A}(s') \right) p_{D}(s, d) - v_{A}(s) \\
\epsilon_{D}(s, d, p_{A}) &= \sum_{a \in \mathcal{A}_{A}(s)} \left(r_{A}(s, a, d) + \gamma \sum_{s' \in \mathbf{S}} p(s'|s, a, d) v_{D}(s') \right) p_{A}(s, a) - v_{D}(s)
\end{aligned} \tag{7.11}$$

Notice that when the policies converge, each $\hat{\epsilon}_{k}(\bar{s}_{t}, \bar{s}_{t+1}, a_{t}, d_{t}) = 0$ for both $k \in \{A, D\}$. In fact $\hat{\epsilon}_{k}(.)$ in Eq. (7.9) is the stochastic approximation of $\epsilon_{k}(.)$ in Eq. (7.11) [111]. This implies that when the update equations converge, there is no Bellman error. In deriving the above update equation, the NLP given in Problem 7.3.3 is converted into the set of sub problems related to each state, $s \in \mathbf{S}$ and each player, $k \in \{A, D\}$. The objective of the set of sub problems is to ensure that there is no Bellman error. Then in update equations, first values for the players at each state is updated using value iteration. Then policies are updated in the decent direction using the gradient estimates of the objective function, $\psi(\mathbf{z})$, to ensure convergence to a set of points that satisfy $p_{A}(s, a)\epsilon_{A}(s, a, p_{D}) = 0$ and $p_{D}(s, d)\epsilon_{D}(s, d, p_{A}) = 0$ for all $a \in \mathcal{A}_{A}(s)$, $d \in \mathcal{A}_{D}(s)$ and $s \in \mathbf{S}$. Also notice that the update equations operate in model-free settings, i.e free of transition probabilities ($p(s'|s, a, d)$). We refer reader to [104] for the proof of convergence of the ON-SGSP algorithm.

7.4 A Reinforcement Learning Algorithm to Compute Equilibrium Policies of Stochastic DIFT-APT Game

In this section we first identify a set of policies, $(\hat{\mathbf{p}}_{A}, \hat{\mathbf{p}}_{D}) \subset (\mathbf{p}_{A}, \mathbf{p}_{D})$, in Γ that will violate Assumption 7.3.4. Then we prove $(\hat{\mathbf{p}}_{A}, \hat{\mathbf{p}}_{D})$ does not form an NE in Γ. We also show that if ON-SGSP algorithm returns a policy in $(\hat{\mathbf{p}}_{A}, \hat{\mathbf{p}}_{D})$ at iteration t, then algorithm will terminate with

such a policy. In order to avoid such policies, $(\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$, we modify ON-SGSP algorithm to solve for the stationary Nash equilibrium policies of the Γ.

First we denote $p_k|_{\hat{\mathbf{S}} \subset \mathbf{S}}$ to be a policy of player k restricted to a set of states $\hat{\mathbf{S}} \subset \mathbf{S}$. Then we identify specific set of stochastic stationary polices in Γ as follows.

Definition 7.4.1. Define the set of policies $(\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D) \subset (\mathbf{p}_A, \mathbf{p}_D)$ that satisfy the following conditions for a set of states $\hat{\mathbf{S}} \subset \mathbf{S}$. For $(\hat{p}_A, \hat{p}_D) \in (\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$

1. $\hat{p}_A|_{\hat{\mathbf{S}} \subset \mathbf{S}}$ induces at-least one cycle in the restricted state space $\hat{\mathbf{S}}$.

2. $\hat{p}_D|_{\hat{\mathbf{S}} \subset \mathbf{S}} = 0_{|\hat{\mathbf{S}}|}$, where $0_{|\hat{\mathbf{S}}|}$ represents the all zeros vector of length $|\hat{\mathbf{S}}|$.

Notice that policy pair $(\hat{p}_A, \hat{p}_D)$ induces a recurrent class consisting of states in $\hat{\mathbf{S}}$ and a transient class with states $s \in \{\mathbf{S} \backslash \hat{\mathbf{S}}\}$. Hence $(\hat{p}_A, \hat{p}_D)$ does not satisfy Assumption 7.3.4. In the next theorem we prove three important properties of a policy pair $(\hat{p}_A, \hat{p}_D)$. We show that $(\hat{p}_A, \hat{p}_D)$ does not form an NE in Γ and it avoids ON-SGSP algorithm in [104] from converging to NE.

Theorem 7.4.2. A policy pair $(\hat{p}_A, \hat{p}_D)$ satisfies the following properties,

1. It does not form an NE in Γ.

2. It avoids ON-SGSP algorithm from converging to an NE.

3. Let $\mathbf{v}_k^t|_{\hat{\mathbf{S}} \subset \mathbf{S}}$ denote a value vector at time instance t for player k restricted to a set of states $\hat{\mathbf{S}} \subset \mathbf{S}$. Then $\mathbf{v}_k^t|_{\hat{\mathbf{S}}} = 0_{|\hat{\mathbf{S}}|}$ for $k \in \{A, D\}$.

Proof. 1. Let p_T denote the probability of $\mathcal{P}_A$ getting detected by $\mathcal{P}_D$. Assume $\mathcal{P}_D$ is performing a security analysis at a state $s \in \hat{\mathbf{S}}$ with arbitrary small probability, i.e., $0 < p_D(s, d = 1) << 1$. Since $\mathcal{P}_A$'s policy is such a way that it forms a cycle among the states $s \in \hat{\mathbf{S}}$, we can write,

$$p_T = \lim_{\hat{t} \to \infty} \sum_{t=0}^{\hat{t}} (1 - p_D(s, d = 1))^t (p_D(s, d = 1))(1 - f_n(s))$$

Notice that with $0 < f_n(s) < 1$, $p_T \to 1$ almost surely. It suggests that $\mathcal{P}_D$ can improve its value for the game by starting to perform security analysis with $0 < p_D(s, d = 1) << 1$ at

any state $s \in \hat{\mathbf{S}}$. Hence it violates that fact that at NE no player can improve their own value of the game by unilaterally deviating from their current policy (Eq. (7.7)).

2. In each iteration t, ON-SGSP algorithm selects actions from the players $\mathcal{P}_A$ and $\mathcal{P}_D$ according to policies at time t, p_A^t and p_D^t, respectively [104]. If these policies belong to the set of policies in $(\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$ then by Definition 7.4.1, these policies will induce at-least one cycle in the state space, which will force ON-SGSP algorithm to update only to a policy in $(\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$. By property 1 these policies do not form an NE.

3. Assume policy pair (p_A^t, p_D^t) has converged to $(\hat{p}_A^t, \hat{p}_D^t) \in (\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$ in the policy updates in Eq. (7.9). Let a cycle induced by $\hat{p}_A^t$ consist of set of states $\hat{\mathbf{S}} := \{s^1, s^2, \ldots, s^l, \ldots, s^{\bar{l}}\}$, where $2 \leq \bar{l} \leq MN$. From policy update equations in Eq. (7.9), convergence implies that $\hat{\epsilon}_A(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t) = \hat{\epsilon}_D(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t) = 0$ for all $\bar{s}_t, \bar{s}_{t+1} \in \hat{\mathbf{S}}$ for the underlying policy pair $(\hat{p}_A^t, \hat{p}_D^t)$. Notice that $r_A(\bar{s}_t, a_t, d_t) = r_D(\bar{s}_t, a_t, d_t) = 0$ for any policy pair belongs to the set $(\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$. Now we can write set of equations, $\gamma v_k^t(s^2) = v_k^t(s^1), \gamma v_k^t(s^3) = v_k^t(s^2), \ldots, \gamma v_k^t(s^1) = v_k^t(s^{\bar{l}})$, for $k \in \{A, D\}$. This suggests either $\gamma = 1$ or $v_k^t(s^1) = v_k^t(s^2) = \ldots, = v_k^t(s^{\bar{l}}) = 0$. But since $0 \leq \gamma < 1$, we conclude that $v_k^t(s^l) = 0$ for all $l = 1, 2, \ldots \bar{l}$.

□

Next we show that any stochastic stationary policies $p_A \notin \hat{\mathbf{p}}_A$ and $p_D \notin \hat{\mathbf{p}}_D$ will fulfill Assumption 7.3.4 under $\mathbf{P}(p_A, p_D)|_{s_0}$. Here, $\mathbf{P}(p_A, p_D)|_{s_0}$ represents the state space induced by the policies (p_A, p_D) restricted to the states that can be only reached from s_0. Note that, since our game, Γ, the initial state at time $t = 1$ is s_0, states induced by the transition matrix $\mathbf{P}(p_A, p_D)|_{s_0}$ is the only positive recurrent class. Hence, we only consider any policy for a player at a state that is reachable from s_0.

Theorem 7.4.3. Any policy pair $(p_A, p_D) \in (\mathbf{p}_A, \mathbf{p}_D)$ and $(p_A, p_D) \notin (\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$ satisfies Assumption 7.3.4 under $\mathbf{P}(p_A, p_D)|_{s_0}$.

Proof. Let $\bar{\mathbf{S}} \subseteq \mathbf{S}$ be the state space induced by $\mathbf{P}(p_A, p_D)|_{s_0}$ for a stochastic stationary policy pair such that $(p_A, p_D) \notin (\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$. Pick two arbitrary states $s, s' \in \bar{\mathbf{S}}$. By the definition of $\mathbf{P}(p_A, p_D)|_{s_0}$, there exists a path from s_0 to s'. Further, now if $p_D(s, d = 0) \neq 1$, then there exists a path from s to a state $s'' = \tau^j$ where j is equal to the stage of state s. If $p_D(s, d = 0) = 1$, then since policy p_A does not create a cycle among any subset of states, $\mathbf{S}' \subseteq \bar{\mathbf{S}}$, satisfying $p_D(\bar{s}, d = 0) = 1$ for $\bar{s} \in \mathbf{S}'$, a path starting from state s will eventually end in a state $s'' \in \bar{\mathbf{S}}$ where $p_D(s'', d = 0) \neq 1$ or $s'' \in \{\{\mathcal{D}_i^M\} \cup \{\phi^j\}\}$ for $i \in \{1, 2, \ldots, N\}$ and $j \in \{1, 2, \ldots, M\}$. Then according to the game model Γ defined in Section 7.2.2 we have a transition probability $p(s_0|s'', a, d) = 1$. This implies there exists a path from s to s' under $\mathbf{P}(p_A, p_D)|_{s_0}$. Since s and s' are two arbitrary states, we conclude that under a policy (p_A, p_D) such that $(p_A, p_D) \notin (\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$, the state space included by $\mathbf{P}(p_A, p_D)|_{s_0}$ is irreducible.

□

Now we present Algorithm 7, modified ON-SGSP algorithm, that provides a pseudo-code for calculating Nash equilibrium of Γ. The key steps of Algorithm 7 is explained here. In order to avoid Algorithm 7 converging to non NE policies, it is important to initialize values and policies such that (v_A^0, v_D^0) where $v_A^0(s) \neq 0$ and $v_D^0(s) \neq 0$ for all $s \in \mathbf{S}$ and (p_A^0, p_D^0) where $(p_A^0, p_D^0) \notin (\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$, respectively (see Theorem 7.4.2 and related discussion therein). Other initializations such as initial gradient estimation values of the objective function $\psi(\mathbf{z})$, (ζ_A^0, ζ_D^0), for $k \in \{A, D\}$ can be initialized to any arbitrary real values. Algorithm 7 starts at the state s_0. Then at each iteration t, both players choose their actions a_t and d_t from $p_A^t(\bar{s}_t)$ and $p_D^t(\bar{s}_t)$, respectively. Next both players observe the next state $\bar{s}_{t+1}$ and their respective rewards $r_A(\bar{s}_t, a_t, d_t)$ and $r_D(\bar{s}_t, a_t, d_t)$. Then players update their discounted values, gradient and policy according to the update equations as in Eq. (7.9).

Algorithm 7 Algorithm to compute Nash equilibrium of Γ

Input: State space ($\mathbf{S}$), step-sizes ($b(n)$, $c(n)$), number of iterations ($T >> 0$), initial value vectors (v_A^0, v_D^0) where $v_A^0(s) \neq 0$ and $v_D^0(s) \neq 0$ for all $s \in \mathbf{S}$, initial gradient values (ζ_A^0, ζ_D^0) and initial policies (p_A^0, p_D^0) where $(p_A^0, p_D^0) \notin (\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$.

Output: Equilibrium policies, $(p_A^\star, p_D^\star) \leftarrow (p_A^T, p_D^T)$ and equilibrium payoff vectors, $(\mathbf{v}_A^\star, \mathbf{v}_D^\star) \leftarrow (v_A^T, v_D^T)$.

Initialization: $t \leftarrow 1, (v_A^t, v_D^t) \leftarrow (v_A^0, v_D^0), (\zeta_A^t, \zeta_D^t) \leftarrow (\zeta_A^0, \zeta_D^0), (p_A^t, p_D^t) \leftarrow (p_A^0, p_D^0), \bar{s}_t \leftarrow s_0$ and $\hat{S} \leftarrow \emptyset$

while $t \leq T$ **do**

  if $\bar{s}_t = s_0$ **then** $\hat{S} \leftarrow \emptyset$

  else

    if $\bar{s}_t \notin \hat{S}$ **then** $\hat{S} = \hat{S} \cup \{\bar{s}_t\}$

    else

      if $\mathbf{v}_A^t|_{\hat{\mathbf{S}}} = \mathbf{v}_D^t|_{\hat{\mathbf{S}}} = 0_{|\hat{\mathbf{S}}|}$ & conditions 1,2 in Definition 7.4.1 hold **then**

        $t \leftarrow 1, \bar{s}_t \leftarrow s_0, (v_A^t|_{\hat{\mathbf{S}}}, v_D^t|_{\hat{\mathbf{S}}}) \leftarrow (v_A^0|_{\hat{\mathbf{S}}}, v_D^0|_{\hat{\mathbf{S}}})$ and $(p_A^t|_{\hat{\mathbf{S}}}, p_D^t|_{\hat{\mathbf{S}}}) \leftarrow (p_A^0|_{\hat{\mathbf{S}}}, p_D^0|_{\hat{\mathbf{S}}})$

      end if

    end if

  end if

  $\mathcal{P}_A$ and $\mathcal{P}_D$ simultaneously play a_t from $p_A^t(\bar{s}_t)$ and d_t from $p_D^t(\bar{s}_t)$ at $\bar{s}_t$

  Next state $\bar{s}_{t+1}$ is revealed

  $\mathcal{P}_A$ and $\mathcal{P}_D$ observes their respective rewards $r_A(\bar{s}_t, a_t, d_t)$ and $r_D(\bar{s}_t, a_t, d_t)$.

  Define

$$\hat{\epsilon}_A(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t) = r_A(\bar{s}_t, a_t, d_t) + \gamma v_A^t(\bar{s}_{t+1}) - v_A^t(\bar{s}_t)$$
$$\hat{\epsilon}_D(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t) = r_D(\bar{s}_t, a_t, d_t) + \gamma v_D^t(\bar{s}_{t+1}) - v_D^t(\bar{s}_t)$$

  Value updates:

$$v_A^{t+1}(\bar{s}_t) = v_A^t(\bar{s}_t) + c(t)[\hat{\epsilon}_A(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)]$$
$$v_D^{t+1}(\bar{s}_t) = v_D^t(\bar{s}_t) + c(t)[\hat{\epsilon}_D(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)]$$

  Gradient estimations:

$$\zeta_A^{t+1}(\bar{s}_t, a_t) = \zeta_A^t(\bar{s}_t, a_t) + c(t)\left[\sum_{k\in\{A,D\}}(\hat{\epsilon}_k(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)) - \zeta_A^t(\bar{s}_t, a_t)\right]$$
$$\zeta_D^{t+1}(\bar{s}_t, d_t) = \zeta_D^t(\bar{s}_t, d_t) + c(t)\left[\sum_{k\in\{A,D\}}(\hat{\epsilon}_k(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)) - \zeta_D^t(\bar{s}_t, d_t)\right]$$

  Policy updates:

$$p_A^{t+1}(\bar{s}_t, a_t) = \Pi\left(p_A^t(\bar{s}_t, a_t) - b(t)\sqrt{p_A^t(\bar{s}_t, a_t)} \times |\hat{\epsilon}_A(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)|\overline{\text{sgn}}(-\zeta_A^{t+1}(\bar{s}_t, a_t))\right)$$
$$p_D^{t+1}(\bar{s}_t, d_t) = \Pi\left(p_D^t(\bar{s}_t, d_t) - b(t)\sqrt{p_D^t(\bar{s}_t, d_t)} \times |\hat{\epsilon}_D(\bar{s}_t, \bar{s}_{t+1}, a_t, d_t)|\overline{\text{sgn}}(-\zeta_D^{t+1}(\bar{s}_t, d_t))\right)$$

  $t \leftarrow t + 1$

end while

During the execution of Algorithm 7, at each iteration t, conditions from line 5 to line 13 is checked to avoid policies in $(\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$. These conditions are based on a set $\hat{S}$, which satisfies conditions 1,2 in Definition 7.4.1 and property 3 in Theorem 7.4.2. Set $\hat{S}$ is set to empty set at each time Algorithm 7 observes $\bar{s}_t = s_0$. Then in each iteration t, algorithm will add the current state $\bar{s}_t$ to the set $\hat{S}$ if it is not present in the set $\hat{S}$. If algorithm notices that $\hat{S}$ already contains $\bar{s}_t$, then at this instance $\hat{S}$ consists of set of states whose occurrence in time form a cycle. In that case we check whether the conditions 1,2 in Definition 7.4.1 and property 3 in Theorem 7.4.2 are satisfied in order to verify if the corresponding policy $(p_A^t, p_D^t) \in (\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$. If all the properties are satisfied, then we reinitialize $(v_A^t|_{\hat{S}}, v_D^t|_{\hat{S}})$ and $(p_A^t|_{\hat{S}}, p_D^t|_{\hat{S}})$ to $(v_A^0|_{\hat{S}}, v_D^0|_{\hat{S}})$ and $(p_A^0|_{\hat{S}}, p_D^0|_{\hat{S}})$, respectively with $t \leftarrow 1$ and $\bar{s}_t \leftarrow s_0$, we make algorithm to re update policies in S such that it will converge to an NE. Since the number of cycles that can be induced in the state space is finite, this will make sure algorithm will eventually avoid all policies in $(\hat{\mathbf{p}}_A, \hat{\mathbf{p}}_D)$ and converge to an NE.

7.5 Simulation Study

In this section, we test Algorithm 7 on a real world nation state attack dataset recorded using Refinable Attack INvestigation (RAIN) architecture [64]. RAIN records system-call events during the runtime of the underlying host computer system. We show that our algorithm successfully converge to an NE of the APT vs. DIFT game, Γ, corresponding to the IFG extracted from the nation state attack dataset for three different values of discounting factors, γ, used.

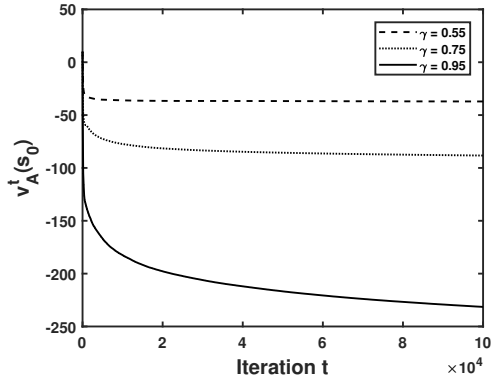
Nation state attack was conducted by the US DARPA red-team during an evaluation of RAIN architecture. The attack runs through three consecutive days but we use only day one system log data for our analysis. Goal of the first day attack is to exfiltrate sensitive information from a company by establishing a back door in a victim computer system. During the first day of attack we observe four attack stages. Adversary enters the system through spear phishing attack and lead the victim to a website that was hosting ads from a malicious web server. Then the adversary exploit a vulnerability in the Firefox browser in first stage of the attack. Next the adversary fingerprints the compromised system to detect running processes and network information during stage two. In stage three adversary writes a malicious program to disk. Lastly in fourth and final stage adversary

establishes a backdoor to continuously exfiltrates the companies sensitive data from the victim computer system.

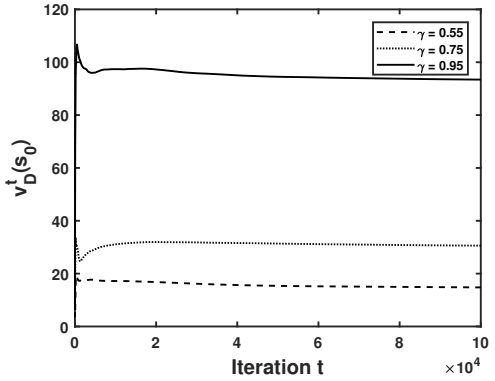
Initial IFG converted from attack data resulted in a coarse-grain graph with $1,32000$ nodes and approximately 2 million edges. Coarse graph captures the whole system data during the recording time which includes the attack related data and lots of data related to the system's background processes (noise). Hence coarse graph provides very little security sensitive (attack related) information about the underlying system. We pruned the coarse graph to extract the security sensitive information about the system from the log data [64]. First step of the pruning includes upstream, downstream and point to point stream techniques presented in [64]. Then we further combine object nodes (e.g., files, net-flow objects) that belong to the same directories or that use the same network socket. The resulting pruned information flow graph consists of 30 nodes $(N = 30)$ out of which 8 nodes are identified as attack destination nodes corresponding to the 4 stages $(M = 4)$ of the day 01 nation state attack. One node related to a net-flow object has been identified as an entry point used for the attack ($|\lambda| = 1$).

Setting up reward parameters for players at each stage depends on the criticality of the resources compromised at each stage. In the nation state attack adversary, $\mathcal{P}_A$, gathers information in the intermediate stages that are critical to achieve the final goal. For example fingerprinting done at stage two of the attack helps adversary to establish a back door to the victim system. In order to capture adversary gaining information needed for his final goal while passing through each stage of the attack, we set reward parameters of the players to $\beta_A^1 = 10, \beta_A^2 = 20, \beta_A^3 = 50, \beta_A^4 = 120$, $\beta_D^1 = -10, \beta_D^2 = -20, \beta_D^3 = -50, \beta_D^4 = -120$, $\alpha_A^j = \sigma_A^j = -200$, for $j \in \{1, \ldots, 4\}$, and $\alpha_D^j = \sigma_D^j = 200$, for $j \in \{1, \ldots, 4\}$.

Figure 7.1 shows convergence of the discounted values, $v_A^t(s_0)$ (Figure 7.1(a)) and $v_D^t(s_0)$ (Figure 7.1(b)), of $\mathcal{P}_A$ and $\mathcal{P}_D$ for three discounting factors $\gamma = 0.55, 0.75$ and 0.95. In all cases considered, Algorithm 7 converges to a NE of Γ in finite time under the IFG of nation state attack dataset and aforementioned reward values. Furthermore, the results show that defender achieves better payoff values at the NE when $\gamma \to 1$. This implies that Algorithm 7 can provide a good estimate on the NE of Γ when $\gamma \to 1$.



(a) Discounted value of $\mathcal{P}_A$ at t^{th} iteration, $v_A^t(s_0)$



(b) Discounted value of $\mathcal{P}_D$ at t^{th} iteration, $v_D^t(s_0)$

Figure 7.1: Convergence of the discounted values, $v_A^t(s_0)$ and $v_D^t(s_0)$, of $\mathcal{P}_A$ and $\mathcal{P}_D$ for three discounting factors $\gamma = 0.55, 0.75$ and 0.95. At $t = 1$, $(v_A^0, v_D^0) = (\mathbf{10}, \mathbf{10})$ was used as initial values for the players and initial player policies, (p_A^0, p_D^0), were set to uniform distributions across actions at each states.

7.6 Summary

Advanced Persistent Threats (APTs) are sophisticated, strategic, and stealthy attacks consisting of multiple stages that undergo continuously over a long period of time to achieve a specific malicious objective. In this chapter, we presented an analytical model that captures the system level interactions between APTs and DIFT for enabling a resource efficient mechanism that can detect and prevent threats imposed by APTs. We modeled the interaction between APT and DIFT as a two-player, nonzero-sum, imperfect information, stochastic game in infinite-horizon. The game consists of multiple stages where each stage corresponds to a stage in the attack. The transition probabilities of the stochastic game, depend on the effectiveness of the defense mechanism (DIFT) such as false-negative rates, are assumed to be unknown. We adopted a model-free reinforcement learning approach and proposed a two-time scale algorithm that converge to a Nash equilibrium (NE) of the APT vs. DIFT stochastic game. The proposed algorithm utilized the structure of the game and is based on the two-time scale algorithm in [104]. In order to evaluate the performance of the proposed method, we conducted experimental analysis of the model and the algorithm on nation state attack data obtained from Refinable Attack INvestigation (RAIN) [64] framework.

Chapter 8

AVERAGE REWARD STOCHASTIC DIFT-APT GAMES WITH REINFORCEMENT LEARNING

8.1 Motivation

In Chapter 6 and Chapter 7, we considered a discounted stochastic game model as it guarantees the existence of a Nash Equilibrium for any information flow graph induced game. The simulation results from Chapter 6 and Chapter 7 show that the payoff of DIFT can be significantly increased when the discount factor $\gamma \rightarrow 1$. This suggests that discounting the stochastic game undermines the long-term behavior of the interactions between DIFT and APT. The long-term behavior of APT permits DIFT more opportunities to perform security analysis on adversarial information flows at multiple system locations. Therefore, long-term behavior of APTs can be exploited by the DIFT to increase the chances of successfully detecting APTs while minimizing the resource cost. However, the results show that the convergence rates become significantly low and convergence error increases when $\gamma \rightarrow 1$. True long-term behavior of APT can be exploited by modeling the interactions as an average reward stochastic game [50]. Hence, in this chapter we first present our average reward stochastic DIFT-APT game and show the existence of an average-reward Nash equilibrium for DIFT-APT game. Then, we develop an actor-critic reinforcement learning algorithm, RL-ARNE, that finds an average reward Nash equilibrium policies of a stochastic game. The contributions of this chapter are the following.

- We model the long-term, stealthy, strategic interactions between DIFT and APT as a two-player, nonzero-sum, average reward, infinite-horizon *stochastic game*. The proposed game model captures the resource costs associated with DIFT in performing security analysis as well as the FPs and FNs of DIFT.

- We propose a solution approach to DIFT-APT game using a reinforcement learning-based algorithm, RL-ARNE, that learns an average reward Nash equilibrium (ARNE) of nonzero-sum stochastic games using TD error minimization.

- We prove the convergence of RL-ARNE algorithm to an ARNE of the game using stochastic approximation.

- We evaluate the performance of our approach via an experimental analysis on ransomware and nation-state attack data obtained from Refinable Attack INvestigation (RAIN) [64].

8.2 Preliminaries on Average Reward Stochastic Games and Stochastic Approximation

Stochastic Games: A stochastic game $\mathbb{G}$ is defined as a tuple $< K, \mathbf{S}, \mathcal{A}, \mathbf{P}, r >$, where K denotes the number of players, $\mathbf{S}$ represents the state space, $\mathcal{A} := \mathcal{A}_1 \times \ldots, \times \mathcal{A}_K$ denotes the action space, $\mathbf{P}$ designates the transition probability kernel, and r represents the reward functions. Here $\mathbf{S}$ and $\mathcal{A}$ are finite spaces. Let $\mathcal{A}_k := \cup_{s \in \mathbf{S}} \mathcal{A}_k(s)$ be the action space of the game corresponding to each player $k \in \{1, \ldots, K\}$, where $\mathcal{A}_k(s)$ denotes the set of actions allowed for player k at state $s \in \mathbf{S}$. Let $\boldsymbol{\pi}_k$ be the set of stationary policies corresponding to player $k \in \{1, \ldots, K\}$ in $\mathbb{G}$. Then a policy $\pi_k \in \boldsymbol{\pi}_k$ is said to be a deterministic stationary policy if $\pi_k \in \{0, 1\}^{|\mathcal{A}_k|}$ and said to be a stochastic stationary policy if $\pi_k \in [0, 1]^{|\mathcal{A}_k|}$. Let $\mathbf{P}(s'|s, a_1, \ldots, a_K)$ be the probability of transitioning from state $s \in \mathbf{S}$ to a state $s' \in \mathbf{S}$ under set of actions $(a_1, \ldots, a_K)$, where $a_k \in \mathcal{A}_k(s)$ denotes the action chosen by player k at the state s. Further let $r_k(s, a_1, \ldots, a_K, s')$ be the reward received by the player k when state of the game transitions from states s to s' under set of actions $(a_1, \ldots, a_K)$ of the players at state s.

Average Reward Payoff Structure: Let $\pi = (\pi_1, \ldots, \pi_K)$. Define $\rho_k(s, \pi)$ to be the average reward payoff of player k when the game starts at an arbitrary state $s \in \mathbf{S}$ and the players follow their respective policies π. Let s^t and a_k^t be the state of game at time t and the action of player k at time t, respectively. Then $\rho_k(s, \pi)$ is defined as

$$\rho_k(s, \pi) = \liminf_{T \to \infty} \frac{1}{T+1} \sum_{t=0}^{T} \mathbb{E}_{s,\pi}[r_k(s^t, a_1^t, \ldots, a_K^t)], \tag{8.1}$$

where the term $\mathbb{E}_{s,\pi}[r_k(s_t, a_1^t, \ldots, a_K^t)]$ denotes the expected reward at time t when the game starts from a state s and the players draw a set of actions $(a_1^t, \ldots, a_K^t)$ at current state s^t based on their respective policies from π. All the players in $\mathbb{G}$ aim to maximize their individual payoff values in Eqn. (8.1). Let $-k$ be the opponents of a player $k \in \{1, \ldots, K\}$ (i.e., $-k := \{1, \ldots, K\} \backslash k$). Then let $\pi_{-k} := \{\pi_1, \ldots, \pi_K\} \setminus \pi_k$ denotes a set of stationary policies of the opponents of player k. Equilibrium of $\mathbb{G}$ under average reward criteria is given below.

Definition 8.2.1 (ARNE). A set of stationary policies $\pi^* = (\pi_1^*, \ldots, \pi_K^*)$ forms an ARNE of $\mathbb{G}$ if and only if $\rho_k(s, \pi_k^*, \pi_{-k}^*) \geq \rho_k(s, \pi_k, \pi_{-k}^*)$, for all $s \in \mathbf{S}, \pi_k \in \boldsymbol{\pi}_k$ and $k \in \{1, \ldots, K\}$.

A policy $\pi^* = (\pi_1^*, \ldots, \pi_K^*)$ is referred to as an ARNE of $\mathbb{G}$. When all the players follow ARNE policy, no player k can increase its payoff by unilaterally deviating from its respective ARNE policy π_k^*.

Unichain Stochastic Games: Let $\mathbf{P}(\pi)$ be the transition probability structure of $\mathbb{G}$ induced by a set of deterministic player policies π. Stochastic games that satisfy Assumption 8.2.2 are referred to as *unichain* stochastic games.

Assumption 8.2.2. Induced Markov chain (MC) $\mathbf{P}(\pi)$ corresponding to every deterministic stationary policy set π contains exactly one recurrent class of states.

Assumption 8.2.2 imposes a structural constraint on the MC induced by *deterministic* stationary policy set. Here, the single recurrent class need not necessarily contain all $s \in \mathbf{S}$. There may exist some transient states in $\mathbf{P}(\pi)$. Also note that any $\mathbb{G}$ that satisfies Assumption 8.2.2 can have multiple recurrent classes in $\mathbf{P}(\pi)$ under some *stochastic* stationary policy set π.

Let $\mathbb{R}_l$ and $\mathbb{T}$ denote a set of states in the l^{th} recurrent class of $\mathbf{P}(\pi)$ for $l \in \{1, \ldots, L\}$, and a set of transient states in $\mathbf{P}(\pi)$, respectively, where L denote the number of recurrent classes. Proposition 8.2.3 gives results on the average reward values of the states in each $\mathbb{R}_l$ and $\mathbb{T}$.

Proposition 8.2.3 ([50], Section 3.2). The following statements are true for any induced MC $\mathbf{P}(\pi)$ of $\mathbb{G}$.

1. For $l \in \{1, \ldots, L\}$ and for all $s \in \mathbb{R}_l$, $\rho_k(s, \pi) = \rho_k^l$, where each ρ_k^l denotes a real-valued constant.

2. $\rho_k(s,\pi) = \sum_{l=1}^{L} q_l(s)\rho_k^l$, if $s \in \mathbb{T}$, where $q_l(s)$ is the probability of reaching a state in l^{th} recurrent class from s.

1) in Proposition 8.2.3 implies that the average reward payoff of player k takes the same value ρ_k^l for each state in the l^{th} recurrent class. 2) suggests that the average reward payoff of a transient state is a convex combination of the average payoffs corresponding to L recurrent classes $\rho_k^1, \ldots, \rho_k^L$. Proposition 8.2.3 shows that for any $\mathbb{G}$, the average reward payoffs corresponding to each state solely depends on the average reward payoffs of the recurrent classes in $\mathbf{P}(\pi)$.

ARNE in Unichain Stochastic Games: Existence of an ARNE for nonzero-sum stochastic games is open. However, the existence of ARNE is shown for some special classes of stochastic games [50].

Proposition 8.2.4 ([123], Theorem 2). There exists an ARNE for a stochastic game that satisfies Assumption 8.2.2.

Let $\pi_k \in \boldsymbol{\pi}_k$ be expressed as $\pi_k = [\pi_k(s)]_{s\in\mathbf{S}}$, where $\pi_k(s) = [\pi_k(s,a_k)]_{a_k\in\mathcal{A}_k(s)}$. Further let $\bar{a} := (a_1,\ldots,a_K)$ and $a_{-k} := \bar{a}\backslash a_k$. Define $\mathbf{P}(s'|s,a_k,\pi_{-k}) = \sum_{a_{-k}\in\mathcal{A}_{-k}(s)} \mathbf{P}(s'|s,\bar{a})\pi_{-k}(s,a_{-k})$, where $\mathbf{P}(s'|s,\bar{a})$ is the probability of transitioning to a state s' from state s under action set $\bar{a}$. Also let $r_k(s,a_k,\pi_{-k}) = \sum_{s'\in\mathbf{S}} \sum_{a_{-k}\in\mathcal{A}_{-k}(s)} \mathbf{P}(s'|s,\bar{a}) r_k(s,\bar{a},s')\pi_{-k}(s,a_{-k})$, where $r_k(s,\bar{a},s')$ is the reward for player k under action set $\bar{a}$ when a state transitions from s to s'. Then a necessary and sufficient condition for characterizing an ARNE of a stochastic game that satisfies Assumption 8.2.2 is given in the following proposition.

Proposition 8.2.5 ([123], Theorem 4). Under Assumption 8.2.2, a set of stochastic stationary policies $(\pi_1,\ldots,\pi_K)$ forms an ARNE in $\mathbb{G}$ if and only if $(\pi_1,\ldots,\pi_K)$ satisfies,

$$\rho_k(s,\pi) + v_k(s) = r_k(s,a_k,\pi_{-k}) + \sum_{s'\in\mathbb{S}} \mathbb{P}(s'|s,a_k,\pi_{-k}) v_k(s') + \lambda_k^{s,a_k}$$
$$\text{for all } s\in\mathbb{S}, a_k\in\mathcal{A}_k(s), k\in\{1,\ldots K\}, \tag{8.2a}$$

$$\rho_k(s,\pi) - \mu_k^{s,a_k} = \sum_{s' \in \mathbb{S}} \mathbb{P}(s'|s,a_k,\pi_{-k})\rho_k(s')$$
$$\text{for all } s \in \mathbb{S}, a_k \in \mathcal{A}_k(s), k \in \{1,\dots K\}, \tag{8.2b}$$
$$\sum_{k\in\{1,\dots,K\}} \sum_{s\in\mathbb{S}} \sum_{a_k \in \mathcal{A}_k(s)} (\lambda_k^{s,a_k} + \mu_k^{s,a_k})\pi_k(s,a_k) = 0. \tag{8.2c}$$

$$\lambda_k^{s,a_k} \geq 0,\ \mu_k^{s,a_k} \geq 0,\ \pi_k(s,a_k) \geq 0$$
$$\text{for all } s \in \mathbb{S}, a_k \in \mathcal{A}_k(s), k \in \{1,\dots K\}, \tag{8.2d}$$
$$\sum_{a_k \in \mathcal{A}_k(s)} \pi_k(s,a_k) = 1 \text{ for all } s \in \mathbb{S}, k \in \{1,\dots K\}, \tag{8.2e}$$

where $v_k(s)$ is the "value" of the game for player k at $s \in \mathbb{S}$.

Stochastic Approximation Algorithms: Let $h : \mathcal{R}^{m_z} \to \mathcal{R}^{m_z}$ be a continuous function of a set of parameters $z \in \mathcal{R}^{m_z}$. Then Stochastic Approximation (SA) algorithms solve a set of equations of the form $h(z) = 0$ based on the noisy measurements of $h(z)$. The classical SA algorithm takes the following form.

$$z^{n+1} = z^n + \delta_z^n[h(z^n) + w_z^n],\ \text{for } n \geq 0 \tag{8.3}$$

Here, n denotes the iteration index and z^n denote the estimation of z at n^{th} iteration of the algorithm. The terms w_z^n and δ_z^n represent the zero mean measurement noise associated with z^n and the step-size of the algorithm, respectively. Note that the stationary points of Eqn. (8.3) coincide with the solutions of $h(z) = 0$ when the noise term w_z^n is zero. Convergence analysis of SA algorithms requires investigating their associated Ordinary Differential Equations (ODEs). The ODE form of the SA algorithm in Eqn. (8.3) is given by $\dot{z} = h(z)$.

Additionally, the following assumptions on step-size δ_z^n are required to guarantee the convergence of an SA algorithm.

Assumption 8.2.6. δ_z^n satisfies, $\sum_{n=0}^{\infty} \delta_z^n = \infty$ and $\sum_{n=0}^{\infty} (\delta_z^n)^2 < \infty$.

Few examples of δ_z^n that satisfy the conditions given in Assumption 8.2.6 are $\delta_z^n = 1/n$ and $\delta_z^n = 1/n\log(n)$. A convergence result that holds for a more general class of SA algorithms is given below.

Proposition 8.2.7 ([**?**, **?**]). Consider an SA algorithm in the following form defined over a set of parameters $z \in \mathcal{R}^{m_z}$ and a continuous function $h : \mathcal{R}^{m_z} \to \mathcal{R}^{m_z}$.

$$z^{n+1} = \Theta(z^n + \delta_z^n[h(z^n) + w_z^n + \kappa^n]), \text{ for } n \geq 0, \tag{8.4}$$

where Θ is a projection operator that projects each z^n iterates onto a compact and convex set $\Lambda \in \mathcal{R}^{m_z}$ and κ^n denotes a bounded random sequence. Let the ODE associated with the iterate in Eqn. (8.4) is

$$\dot{z} = \bar{\Theta}(h(z)), \tag{8.5}$$

where $\bar{\Theta}(h(z)) = \lim_{\eta \to 0} \frac{\Theta(z+\eta h(z)) - z}{\eta}$ and $\bar{\Theta}$ denotes a projection operator that restricts the evolution of ODE in Eqn. (8.5) to the set Λ. Let the nonempty compact set Z denotes a set of asymptotically stable equilibrium points of Eqn. (8.5).

Then z^n converges almost surely to a point in Z as $n \to \infty$ given the following conditions are satisfied,

1. δ_z^n satisfies the conditions in Assumption 8.2.6.

2. $\lim_{n \to \infty} \left(\sup_{\bar{n} > n} \left| \sum_{l=n}^{\bar{n}} \delta_z^n w_z^n \right| \right) = 0$ almost surely.

3. $\lim_{n \to \infty} \kappa^n = 0$ almost surely.

Consider a class of SA algorithms that consist of two interdependent iterates that update on two different time scales (i.e., step-sizes of two iterates are different in the order of magnitude). Let $x \in \mathcal{R}^{m_x}$ and $y \in \mathcal{R}^{m_y}$ and $n \geq 0$. Then the iterates given in the following equations portray a format of such two-time scale SA algorithm.

$$x^{n+1} = x^n + \delta_x^n[f(x^n, y^n) + w_x^n], \; y^{n+1} = y^n + \delta_y^n[g(x^n, y^n) + w_y^n]. \tag{8.6}$$

The following proposition provides a convergence result related to the aforementioned two-time scale SA algorithm.

Proposition 8.2.8 ([**?**], Chapter 6). Consider x^n and y^n iterates given in (8.6). Then, given the iterates in (8.6) are bounded, $\{(x^t, y^t)\}$ converges to $(\psi(y^*), y^*)$ almost surely under the following conditions.

(I) $f : \mathcal{R}^{m_x+m_y} \to \mathcal{R}^{m_x}$ and $g : \mathcal{R}^{m_x+m_y} \to \mathcal{R}^{m_y}$ are Lipschitz.

(II) Iterates x^n and y^n are bounded.

(III) Let $\psi : y \to x$. For all $y \in \mathcal{R}^{m_y}$, $\dot{x} = f(x, y)$ has an asymptotically stable critical point $\psi(y)$ such that function ψ is Lipschitz.

(IV) $\dot{y} = g(\psi(y), y)$ has a global asymptotically stable critical point.

(V) Let ξ^n be an increasing σ-field defined by $\xi^n := \sigma(x^n, \ldots, x^0, y^n, \ldots, y^0, w_x^{n-1}, \ldots, w_x^0, w_y^{n-1}, \ldots, w_y^0)$. Further let κ_x and κ_y be two positive constants. Then w_x^n and w_y^n are two noise sequences that satisfy, $\mathbb{E}[w_x^n|\xi^n] = 0$, $\mathbb{E}[w_y^n|\xi^n] = 0$, $\mathbb{E}[\| w_x^n \|^2 |\xi^n] \leq \kappa_x(1+ \| x^n \| + \| y^n \|)$, and $\mathbb{E}[\| w_y^n \|^2 |\xi^n] \leq \kappa_y(1+ \| x^n \| + \| y^n \|)$.

(VI) δ_x^n and δ_y^n satisfy Assumption 8.2.6. Additionally, $\lim_{n\to\infty} \sup \frac{\delta_y^n}{\delta_x^n} = 0$.

8.3 Stochastic DIFT-APT Game Formulation

In this section, we model the interactions between a DIFT-based defender (D) and an APT adversary (A) as a two-player stochastic game (DIFT-APT game). The DIFT-APT game unfolds in the infinite time horizon $t \in \mathcal{T} := \{1, 2, \ldots\}$.

8.3.1 System and Defender Models

System Model: The computer system is abstracted through Information Flow Graph (IFG), $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$, where the set of nodes, $V_\mathcal{G} = \{u_1, \ldots, u_N\}$ depicts the N distinct components of the computer and the set of edges $E_\mathcal{G} \subset V_\mathcal{G} \times V_\mathcal{G}$ represents the feasibility of transferring information flows between the components. Specifically, an edge $e_{ij} \in E_\mathcal{G}$ indicates that an information flow

can be transferred from a component u_i to another component u_j, where $i, j \in \{1, \ldots, N\}$ and $i \neq j$. Let $\mathcal{E} \subset V_{\mathcal{G}}$ be the set of entry points used by APT to infiltrate the computer system. Consider an APT attack that consists of M attack stages and let $\mathcal{D}_j \subset V_{\mathcal{G}}$ for each $j \in \{1, \ldots, M\}$ be the set of components that are targeted by the APT in the j^{th} attack stage. Let $\mathcal{D}_j$ be the set of *destinations* of stage j.

DIFT Defender Model: DIFT tags/taints all the information flows originating from the set of entry points as suspicious flows. Then DIFT tracks the propagation of the tainted flows through the system and initiates security analysis at specific components of the system to detect the APT. Performing security analysis incurs memory and performance overheads to the system which varies across the system components. The objective of DIFT is to select a set of system components for performing security analysis while minimizing the memory and performance overhead. On the other hand, the objective of APT is to evade detection by DIFT and successfully complete the attack by sequentially reaching at least one node from each set $\mathcal{D}_j$, for all $j = 1, \ldots, M$.

8.3.2 State Space and Action Space

Let $\mathbf{S} := \{s_0\} \cup \{V_{\mathcal{G}} \times \{1, \ldots, j\}\} = \{s_0, s_1^j, \ldots, s_N^j\}$, for all $j \in \{1, \ldots, M\}$, represent the finite state space of DIFT-APT game. The state s_0 represents the reconnaissance stage of the attack where APT chooses an entry point of the system to launch the attack. Therefore, at time $t = 0$, DIFT-APT game starts from s_0. A state s_i^j denotes a tagged information flow at a system component $u_i \in V_{\mathcal{G}}$ corresponding to the j^{th} attack stage. Also, note that a state s_i^j, where $u_i \in \mathcal{D}_j$ and $j \in \{1, \ldots, M-1\}$, is associated with APT achieving the intermediate goal of stage j. Moreover, a state s_i^M, where $u_i \in \mathcal{D}_M$, represents APT achieving the final goal of the attack.

Let $\mathcal{N}(s)$ be the set of out-neighboring states of state $s \in \mathbf{S}$. Let $\mathcal{A}_k = \cup_{s \in \mathbf{S}} \mathcal{A}_k(s)$ be the action space of the player $k \in \{D, A\}$, where $\mathcal{A}_k(s)$ denotes the set of actions allowed for player k at a state s. The action sets of the players at any state $s \in \mathbf{S}$ is given by $\mathcal{A}_{D}(s) \in \mathcal{N}(s) \cup \{0\}$ and $\mathcal{A}_{A}(s) \in \mathcal{N}(s) \cup \{\varnothing\}$. Here, $\mathcal{A}_{D}(s) \in \mathcal{N}(s)$ and $\mathcal{A}_{D}(s) = 0$ denote DIFT deciding to perform security analysis at an out-neighboring state and deciding not to perform security analysis,

respectively. Also, $\mathcal{A}_A(s) \in \mathcal{N}(s)$ represents APT deciding to transition to an out-neighboring state of $s \in \mathbf{S}$ and $\varnothing$ represents APT quitting the attack. At each step of the game DIFT and APT *simultaneously* choose their respective actions.

Specifically, there are four cases. (i) $s = s_0$, $\mathcal{A}_A(s) = \{s_i^1 : u_i \in \mathcal{E}\}$ and $\mathcal{A}_D(s) = 0$. Here, APT selects an entry point in the system to initiate the attack. (ii) $\{s = s_i^j : u_i \notin \mathcal{D}_j, j = 1, \ldots, M\}$, $\mathcal{A}_A(s) = \{s_{i'}^j : (u_i, u_{i'}) \in E_{\mathcal{G}}\} \cup \{\varnothing\}$ and $\mathcal{A}_D(s) \in \{s_{i'}^j : (u_i, u_{i'}) \in E_{\mathcal{G}}\} \cup \{0\}$. In other words, APT chooses to transition to one of the out-neighboring node of u_i in stage j or decides to quit the attack ($\varnothing$) and DIFT decides to perform security analysis at an out-neighboring node of u_i in stage j or not. (iii) $\{s = s_i^j : u_i \in \mathcal{D}_j, j = 1, \ldots, M-1\}$, $\mathcal{A}_A(s) = s_i^{j+1}$ and $\mathcal{A}_D(s) = 0$. That is, APT traverses from stage j of the attack to stage $j+1$ and DIFT does not perform a security analysis. (iv) $\{s = s_i^M : s_i \in \mathcal{D}_M\}$. Then, $\mathcal{A}_A(s) = s_0$ which captures the persistency of the APT attack and $\mathcal{A}_D(s) = 0$.

Note that DIFT does not perform security analysis at the states corresponding to s_0 and destinations due to the following reasons. At the entry points there are not enough traces to perform security analysis as attack originates at these system components. The destinations $\mathcal{D}_j$, for $j \in \{1, \ldots, M\}$, typically consist of busy processes and/or confidential files with restricted access. Performing security analysis at states corresponding to entry points and destinations is not allowed.

8.3.3 Policies and Transition Structure

Let s_t be the state of the game at time $t \in \mathcal{T}$. Consider *stationary* policies for DIFT and APT, i.e., decisions made at a state $s_t \in \mathbf{S}$ at any time t only depends on s_t. Let $\boldsymbol{\pi}_D$ and $\boldsymbol{\pi}_A$ be the set of stationary policies of DIFT and APT, respectively. Then stochastic stationary policies of DIFT and APT are defined by $\pi_k \in [0,1]^{|\mathcal{A}_k|}$, where $\pi_k \in \boldsymbol{\pi}_k$ and $k \in \{D, A\}$. Moreover, let $\pi_k = [\pi_k(s)]_{s \in \mathbf{S}}$ and $\pi_k(s) = [\pi_k(s, a_k)]_{a_k \in \mathcal{A}_k(s)}$, where $\pi_k(s)$ and $\pi_k(s, a_k)$ denote the policy of a player $k \in \{D, A\}$ at a state $s \in \mathbf{S}$ and probability of player k choosing an action $a_k \in \mathcal{A}_k(s)$ at the state s. In what follows, we use $a_k = d$ when $k = D$ and $a_k = a$ when $k = A$ to denote an action of DIFT and APT at a state s, respectively.

Assume state transitions are *stationary*, i.e., state at time $t+1$, s_{t+1} depends only on the current

state s_t and the actions a_t and d_t of both players at the state s_t, for any $t \in \mathcal{T}$. Let $\mathbf{P}$ be the transition structure of the DIFT-APT game. Then $\mathbf{P}(\pi_D, \pi_A)$ represents the state transition matrix of the game resulting from $(\pi_D, \pi_A) \in (\boldsymbol{\pi}_D, \boldsymbol{\pi}_A)$. Then,

$$\mathbf{P}(\pi_D, \pi_A) = [\mathbf{P}(s'|s, \pi_D, \pi_A)]_{s,s' \in \mathbf{S}}, \text{ where}$$

$$\mathbf{P}(s'|s, \pi_D, \pi_A) = \sum_{d \in \mathcal{A}_D(s)} \sum_{a \in \mathcal{A}_A(s)} \mathbf{P}(s'|s, d, a) \pi_D(s, d) \pi_A(s, a). \tag{8.7}$$

Here $\mathbf{P}(s'|s, d, a)$ denotes the probability of transitioning to state s' from state s when DIFT chooses an action $d \in \mathcal{A}_D(s)$ and APT chooses an action $a \in \mathcal{A}_A(s)$. Let $FN(s_i^j)$ denote the rate of FNs generated at a system component $u_i \in V_G$ while analyzing a tagged flow corresponding to stage j of the attack. Then for a state s_t, actions d_t and a_t the possible next state s_{t+1} are as follows,

$$s_{t+1} = \begin{cases} s_i^j, & \text{w.p } 1, & \text{when } d_t = 0 \text{ and } a_t = s_i^j \\ s_i^j, & \text{w.p } FN(s_i^j), & \text{when } d_t = a_t = s_i^j \\ s_0, & \text{w.p } 1 - FN(s_i^j), & \text{when } d_t = a_t = s_i^j \\ s_i^j, & \text{w.p } 1, & \text{when } d_t \neq a_t \\ s_0, & \text{w.p } 1, & \text{when } a_t = \varnothing. \end{cases} \tag{8.8}$$

In the first case of Eqn. (8.8), the next state of the game is uniquely defined by the action of APT as DIFT does not perform security analysis. In the second and thrid cases of Eqn. (8.8), DIFT decides correctly to perform security analysis on the malicious flow. Note that the security analysis of DIFT can not accurately detect a possible attack due to generation of FNs. Hence the next state of the game is determined by the action of APT (in case two) when a FN is generated. And the next state of the game is s_0 (in case three) when APT is detected by DIFT and APT starts a new attack. Case four of Eqn. (8.8) represents DIFT performing security analysis on a benign flow. In such a case, the state of the game is uniquely defined by the action of the adversary. Finally, in case five of Eqn. (8.8), i.e., when APT decides to quit the attack, the next state of the game is the initial state s_0.

FNs of the DIFT scheme arise from the limitations of the security rules that can be deployed at each node of the IFG (i.e., processes and objects in the system). Such limitations are due to

variations in the number of rules and the depth of the security analysis[1] (e.g., system call level trace, CPU instruction level trace) that can be implemented at each node of the IFG resulting from the resource constraints including memory, storage and processing power imposed by the system on each IFG node.

8.3.4 Reward Structure

Let $r_{_D}(s, \pi_{_D}, \pi_{_A})$ and $r_{_A}(s, \pi_{_D}, \pi_{_A})$ be the expected reward of DIFT and APT at a state $s \in \mathbf{S}$ under policy pair $(\pi_{_D}, \pi_{_A}) \in (\boldsymbol{\pi}_{_D}, \boldsymbol{\pi}_{_A})$. Then for each $k \in \{D, A\}$,

$$r_k(s, \pi_{_D}, \pi_{_A}) = \sum_{s' \in \mathbf{S}} \sum_{\substack{a \in \mathcal{A}_{_A}(s) \\ d \in \mathcal{A}_{_D}(s)}} \mathbf{P}(s'|s, d, a) \pi_{_D}(s, d) \pi_{_A}(s, a) r_k(s, d, a, s'),$$

where $r_k(s, d, a, s')$ denotes the reward of player k when state transition from s to s' under actions $d \in \mathcal{A}_{_D}(s)$ and $a \in \mathcal{A}_{_A}(s)$ of DIFT and APT, respectively. Moreover, $r_{_D}(s, d, a, s')$ and $r_{_A}(s, d, a, s')$ are defined as follows.

$$r_{_D}(s, d, a, s') = \begin{cases} \alpha_{_D}^j + \mathcal{C}_{_D}(s) & \text{if } d = a,\ s' = s_0 \\ \beta_{_D}^j & \text{if } d = 0,\ s' \in \{s_i^j : u_i \in \mathcal{D}_j\} \\ \sigma_{_D}^j + \mathcal{C}_{_D}(s) & \text{if } d \neq 0,\ a = \varnothing \\ \sigma_{_D}^j & \text{if } d = 0,\ a = \varnothing \\ \mathcal{C}_{_D}(s) & \text{if } d \neq a \text{ and } d \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$r_{_A}(s, d, a, s') = \begin{cases} \alpha_{_A}^j & \text{if } d = a,\ s' = s_0 \\ \beta_{_A}^j & \text{if } s' \in \{s_i^j : u_i \in \mathcal{D}_j\} \\ \sigma_{_A}^j & \text{if } a = \varnothing \\ 0 & \text{otherwise} \end{cases}$$

[1] Detecting an unauthorized use of tagged flow crucially depends on the path traversed by the information flow [41, 124].

The reward structure $r_{_D}(s, d, a, s')$ captures the cost of FPs generation by assigning a cost $\mathcal{C}_{_D}(s)$ whenever $d \neq a$ such that $d \neq 0$. Note that, $r_{_D}(s, d, a, s')$ consists of four components (i) reward $\alpha_D^j > 0$ for DIFT detecting the APT in j^{th} stage (ii) penalty $\beta_D^j < 0$ for APT reaching a destination of stage j, for $j = 1, \ldots, M$ (iii) reward $\sigma_D^j > 0$ for APT quitting the attack in j^{th} stage and (iv) a security cost $\mathcal{C}_{_D}(s) < 0$ that captures the memory and storage costs associated with performing a security checks on a tagged flow at a state $s \in \{s_i^j : u_i \notin \mathcal{D}_j \cup \mathcal{E}\}$. Reward $r_{_A}(s, d, a, s')$ consists of three components (i) penalty $\alpha_A^j < 0$ if APT is detected by DIFT in the j^{th} stage (ii) reward $\beta_A^j > 0$ for APT reaching a destination of stage j, for $j = 1, \ldots, M$ and (iii) penalty $\sigma_A^j < 0$ for APT quitting the attack in j^{th} stage. Since it is not necessary that $r_{_D}(s, d, a, s') = -r_{_A}(s, d, a, s')$ for all $d \in \mathcal{A}_{_D}(s)$, $a \in \mathcal{A}_{_A}(s)$ and $s, s' \in \mathbf{S}$, DIFT-APT game is a *nonzero-sum* game.

8.3.5 Information Structure

Both DIFT and APT are assumed to know the current state, s_t of the game, both action sets $\mathcal{A}_{_D}(s_t)$ and $\mathcal{A}_{_A}(s_t)$, and payoff structure of the DIFT-APT game. But DIFT is unaware whether a tagged flow at s_t is malicious or not and APT does not know the chances of getting detected at s_t. This results in an information *asymmetry* between the players. Hence DIFT-APT game is an *imperfect* information game. Furthermore, both players are unaware of the transition structure $\mathbf{P}$ which depend on the rate of FNs generated at the different states s_t (Eq. (8.8)). Consequently, the DIFT-APT game is an *incomplete* information game.

8.3.6 Solution Concept: ARNE

APTs are stealthy attackers whose interactions with the system span over a long period of time. Hence, players D and A must consider the rewards they incur over the long-term time horizon when they decide on their policies π_D and π_A, respectively. Therefore, average reward payoff criteria is used to evaluate the outcome of DIFT-APT game for a given policy pair $(\pi_D, \pi_A) \in (\boldsymbol{\pi}_D, \boldsymbol{\pi}_A)$. Note that, the DIFT-APT game originates at s_0. Thus the average payoff for player $k \in \{D, A\}$

with policy pair (π_D, π_A) is defined as follows.

$$\rho_k(s_0, \pi_D, \pi_A) = \liminf_{T\to\infty} \frac{1}{T+1} \sum_{t=0}^{T} \mathbb{E}_{s_0,\pi_D,\pi_A}[r_k(s_t, d_t, a_t)].$$

Moreover, a pair of stationary policies (π_D^*, π_A^*) forms an ARNE of DIFT-APT game if and only if

$$\rho_D(s, \pi_D^*, \pi_A^*) \geq \rho_D(s, \pi_D, \pi_A^*), \quad \rho_A(s, \pi_D^*, \pi_A^*) \geq \rho_A(s, \pi_D^*, \pi_A)$$

for all $s \in \mathbf{S}, \pi_k \in \boldsymbol{\pi}_k$.

8.4 Analyzing ARNE of the DIFT-APT Game

In this section we first show the existence of ARNE in DIFT-APT game. Then we provide necessary and sufficient conditions required to characterize an ARNE of DIFT-APT game. Henceforth we assume the following assumption holds for the IFG associated with the DIFT-APT game.

Assumption 8.4.1. The IFG is acyclic.

Any IFG with set of cycles can be converted into an acyclic IFG without loosing any causal relationships between the components given in the original IFG. One such dependency preserving conversion is *node versioning* given in [**?**]. Hence this assumption is not restrictive. Let $\mathbf{P}(\pi_D, \pi_A)$ be the MC induced by a policy pair (π_D, π_A). The following theorem presents properties of DIFT-APT game under Assumption 8.4.1.

Theorem 8.4.2. Let the DIFT-APT game satisfies Assumption 8.4.1. Then, the following properties hold.

1. $\mathbf{P}(\pi_D, \pi_A)$ corresponding to any $(\pi_D, \pi_A) \in (\boldsymbol{\pi}_D, \boldsymbol{\pi}_A)$ consists of a single recurrent class of states (with possibly some transient states reaching the recurrent class).

2. The recurrent class of $\mathbf{P}(\pi_D, \pi_A)$ includes the state s_0.

Proof. Consider a partitioning of the state space such that $\mathbf{S} = \mathbf{S}_1 \cup \mathbf{S}_2$ and $\mathbf{S}_1 \cap \mathbf{S}_2 = \varnothing$. Here $\mathbf{S}_1$ denotes the set of states that are reachable[2] from state s_0 and $\mathbf{S}_2$ denotes the set of states that

[2]In a directed graph a state u is said to be reachable from state v, if there exists a directed path from v to u.

are not reachable from s_0. We prove $1)$ and $2)$ by showing that $\mathbf{S}_1$ forms a single recurrent class of $\mathbf{P}(\pi_D, \pi_A)$ and $\mathbf{S}_2$ forms the set of transient states.

We first show that in $\mathbf{P}(\pi_D, \pi_A)$, state s_0 is reachable from any arbitrary state $s \in \mathbf{S} \setminus \{s_0\}$. The proof consists of two steps. First consider a state $s = s_i^j$, such that $u_i \notin \mathcal{D}_M$ with $j = M$. In other words, the state s is not a state that is corresponding to a final goal of the attack. Let s' be an out neighbor of s. Then s' satisfies one of the two cases. i) $s' = s_0$ and ii) $s' = s_{i'}^{j'} \in \mathbf{S} \setminus \{s_0\}$. Case i) happens if the APT decides to dropout from the game or if DIFT successfully detects the APT. Thus in case i) s_0 is reachable from s.

Case ii) happens when DIFT does not detect APT and the APT chooses to move to an out neighboring state s'. By recursively applying cases i) and ii) at s', we get s_0 is reachable when case i) occurs at least once. What is remaining to show is when only case ii) occurs. In such a case, transitions from s' will eventually reach a state corresponding to a final goal of the attack, i.e., s_i^M with $u_i \in \mathcal{D}_M$, due to the acyclic nature of the IFG imposed by Assumption 8.4.1. Note that at s_i^M with $u_i \in \mathcal{D}_M$ the only transition possible is to s_0. This proves that s_0 is reachable from any state $s \in \mathbf{S} \setminus s_0$.

This along with the definition of $\mathbf{S}_1$ implies that $\mathbf{S}_1$ forms a recurrent class of $\mathbf{P}(\pi_D, \pi_A)$. Also as s_0 is reachable from any state in $\mathbf{S}_2$ and by the definition of $\mathbf{S}_2$, $\mathbf{S}_2$ is the set of transient states. This completes the proof. □

Corollary 8.4.3 below presents the existence of an ARNE in DIFT-APT using Theorem 8.4.2.

Corollary 8.4.3. Let the DIFT-APT game satisfies Assumption 8.4.1. Then, there exits an ARNE for the DIFT-APT game.

Proof. From the condition 1) in Theorem 8.4.2 the DIFT-APT game has a single recurrent class of states in $\mathbf{P}(\pi_D, \pi_A)$ corresponding to any policy pair (π_D, π_A). As a result Assumption 8.2.2 holds for DIFT-APT game. Therefore by Proposition 8.2.4 there exits an ARNE in DIFT-APT game. □

The corollary below gives a necessary and sufficient condition for characterizing an ARNE of DIFT-APT game. Our algorithm for computing ARNE is based on this condition.

Corollary 8.4.4. The following conditions characterizes the ARNE of DIFT-APT game.

$$\rho_k + v_k(s) \geq r_k(s, a_k, \pi_{-k}) + \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, a_k, \pi_{-k}) v_k(s'), \tag{8.9a}$$

$$\sum_{k \in \{D,A\}} \sum_{s \in \mathbf{S}} \sum_{a_k \in \mathcal{A}_k(s)} \Big(\rho_k + v_k(s) - r_k(s, a_k, \pi_{-k}) - \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, a_k, \pi_{-k}) v_k(s') \Big) \pi_k(s, a_k) = 0, \tag{8.9b}$$

$$\sum_{a_k \in \mathcal{A}_k(s)} \pi_k(s, a_k) = 1, \quad \pi_k(s, a_k) \geq 0, \tag{8.9c}$$

where ρ_k denotes the average reward value of player k independent of initial state of the game.

Proof. By Proposition 8.2.5, ARNE of an unichain stochastic game is characterized by conditions (8.2a)-(8.2e). The condition (8.2a) reduce to (8.9a) by substituting $\lambda_k^{s,a_k} \geq 0$ from condition (8.2d). Below is the argument for condition (8.2b).

From Theorem 8.4.2, the MC induced by (π_D, π_A), $\mathbf{P}(\pi_D, \pi_A)$, contains only a single recurrent class. As a consequence, from Proposition 8.2.3, $\rho_k(s, \pi) = \rho_k$ for all $s \in \mathbf{S}$ and $k \in \{D, A\}$. Thus condition (8.2b) in Proposition 8.2.5 reduces to

$$\rho_k - \mu_k^{s,a_k} = \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, a_k, \pi_{-k}) \rho_k = \rho_k \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, a_k, \pi_{-k}) = \rho_k$$

Thus, $\mu_k^{s,a_k} = 0$. Since $\rho_k(s, \pi) = \rho_k$, condition (8.2a) in Proposition 8.2.5 becomes

$$\lambda_k^{s,a_k} = \rho_k + v_k(s) - r_k(s, a_k, \pi_{-k}) - \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, a_k, \pi_{-k}) v_k(s'). \tag{8.10}$$

By substituting $\mu_k^{s,a_k} = 0$ and λ_k^{s,a_k} from Eqn. (8.10), condition (8.2c) reduces to (8.9b). Finally, conditions (8.2d) and (8.2e) together reduce to (8.9c). Thus conditions (8.9a)-(8.9c) characterizes an ARNE in DIFT-APT game. □

8.5 RL-ARNE: A Reinforcement Learning Algorithm to Compute Equilibrium Policies of Average Reward Stochastic Games

In this section we present a RL algorithm that learns ARNE in DIFT-APT game. This algorithm extends to any N-player uni-chain average reward stochastic games.

8.5.1 RL-ARNE Algorithm

Algorithm 8 presents the pseudocode of RL-ARNE, an actor-critic algorithm that computes an ARNE in DIFT-APT game. Algorithm 8 takes the state space ($\mathbf{S}$), transition structure ($\mathbf{P}$), and rewards (r_D and r_A) of DIFT-APT game along with the pre-specified number of iterations ($I \gg 0$) as the inputs and outputs the ARNE policies of DIFT and APT, $(\pi_D^\star, \pi_A^\star)$. Below we detail the steps of Algorithm 8.

Algorithm 8 RL-ARNE Algorithm of DIFT-APT game

Input: State space ($\mathbf{S}$), transition structure ($\mathbf{P}$), rewards (r_D and r_A), number of iterations ($I >> 0$)

Output: ARNE policies, $(\pi_D^\star, \pi_A^\star) \leftarrow (\boldsymbol{\pi}_D^I, \boldsymbol{\pi}_A^I)$

Initialization: $n \leftarrow 0$, $v_k^0 \leftarrow 0$, $\rho_k^0 \leftarrow 0$, $\epsilon_k^0 \leftarrow 0$, $\pi_k^0 \leftarrow \boldsymbol{\pi}_k$ for $k \in \{D, A\}$ and $s \leftarrow s_0$.

while $n \leqslant I$ **do**

  Draw d from $\pi_D^n(s)$ and a from $\pi_A^n(s)$

  Reveal the next state s' according to $\mathbf{P}$

  Observe the rewards $r_D(s, d, a, s')$ and $r_A(s, d, a, s')$

  for $k \in \{D, A\}$ **do**

    $v_k^{n+1}(s) = v_k^n(s) + \delta_v^n[r_k(s, d, a, s') - \rho_k^n + v_k^n(s') - v_k^n(s)]$

    $\rho_k^{n+1} = \rho_k^n + \delta_\rho^n\Big[\frac{n\rho_k^n + r_k(s,d,a,s')}{n+1} - \rho_k^n\Big]$

    $\epsilon_k^{n+1}(s, a_k) = \epsilon_k^n(s, a_k) + \delta_\epsilon^n\big[\sum_{k\in\{D,A\}}(r_k(s, d, a, s') - \rho_k^n + v_k^n(s') - v_k^n(s)) - \epsilon_k^n(s, a_k)\big]$

    $\pi_k^{n+1}(s, a_k) = \Gamma(\pi_k^n(s, a_k) - \delta_\pi^n\sqrt{\pi_k^n(s, a_k)}\big|r_k(s, d, a, s') - \rho_k^n + v_k^n(s') - v_k^n(s)\big|\text{sgn}(-\epsilon_k^n(s, a_k)))$

  end for

  Update the state of DIFT-APT game: $s \leftarrow s'$

  $n \leftarrow n + 1$

end while

Algorithm 8 first initializes the iteration counter n, value functions v_k^n, average rewards ρ_k^0, and the gradient estimates ϵ_k^0 of DIFT and APT (i.e., $k \in \{D, A\}$) to zero. We initialize the policies of DIFT and APT (π_D^0, π_A^0) to have uniform distributions over their respective actions at any state $s \in \mathbf{S}$ and set initial state to s_0, the state that corresponds to the reconnaissance stage of the attack.

At each iteration n, Algorithm 8 first draws the actions of DIFT and APT, d and a according to their respective policies $\pi_D^n(s)$ and $\pi_A^n(s)$ at the current state s of the game (line 5). Then given the triplet (s, d, a), algorithm finds the next state s' of the game according the transition structure $\mathbf{P}$ that models the FPs and FNs of the DIFT (line 6). Next using the quadruplet (s, d, a, s'), algorithm computes the rewards $r_D(s, d, a, s')$ and $r_A(s, d, a, s')$ of DIFT and APT, respectively (line 7).

Iterates in lines 9 compute the value functions $v_k^n(s)$ of the two players, DIFT and APT at the current state $s \in \mathbf{S}$. Algorithm computes the average rewards ρ_k^n of DIFT and APT in line 10. The iterates, $\epsilon_k^n(s, a_k)$ in line 11 compute the gradient of the expression in condition (5b) in Corollary 8.4.4 with respect to the each player's policy $\pi_k^n(s, a_k)$. Then at line 12, using sign gradient descent approach, algorithm updates the policies of each player to minimize the expression in condition (5b) in Corollary 8.4.4. The map Γ projects the policies to probability simplex defined by condition (8.9c) in Corollary 8.4.4. Here, $|\cdot|$ denotes the absolute value. The function $\text{sgn}(\chi)$ denotes the continuous version of the standard sign function (e.g., $\text{sgn}(\chi) = \tanh(c\chi)$ for any constant $c > 1$).

Note that the value updates at Line 9 satisfies the condition (5a) in Corollary 8.4.4. Gradient descent updates of the policies at line 12 satisfy condition (5b) in Corollary 8.4.4. The map Γ ensures that the $\pi_k^n(s, a_k)$ iterates are valid probability distributions by projecting onto the simplex space. Thus by Corollary 8.4.4 Algorithm 8 converges to an ARNE of DIFT-APT games.

The value function (actor) iterates in line 9 and the gradient estimate iterates in line 11 of Algorithm 8 update in a same faster time scale δ_v^n and δ_ϵ^n, respectively. Policy (critic) iterates in line 12 update in a slower time scale δ_π^n. Average reward payoff iterates in line 10 update in an intermediate time scale δ_ρ^n. Hence the step-sizes of the proposed algorithm are chosen such that $\delta_v^n = \delta_\epsilon^n \gg \delta_\rho^n \gg \delta_\pi^n$.

Remark 8.5.1. Note that RL-ARNE algorithm presented in Algorithm VI.1 must be trained offline

due to the information exchange that is required at line 11 of the algorithm. Here, players are required to exchange the information about their respective temporal difference error estimates, $\tilde{\phi}_k(\rho_k^n, v_k^n) = r_k(s, d, a, s') - \rho_k^n + v_k^n(s') - v_k^n(s)$, as the iterates on each player's gradient estimation includes the term $\sum_{k\in\{D,A\}} \tilde{\phi}_k(\rho_k^n, v_k^n)$. Since RL-ARNE algorithm is trained offline and the policies found at the end of the training only depend on their respective actions, players do not require any information exchange on their respective actions when they execute their learned policies in real-time.

Remark 8.5.2. If APT gets detected or quits the attack, it results in reward for the defender and penalty for the attacker. If APT evades detection, it results in penalty for the defender and reward for the attacker. Hence, defender can set $\alpha_A^j = -\alpha_D^j$, $\beta_A^j = -\beta_D^j$, $\sigma_A^j = -\sigma_D^j$. Moreover, we consider a strong attacker model where attacker knows the specific values of the defender's payoff function parameters.

Proposition 8.5.3. Let K, A, and S denote the number of players, the maximum cardinality of the action space of any player, and the cardinality of state space, respectively. Then RL-ARNE in Algorithm 8 has per iteration computation complexity of $O(KA)$ and memory complexity of $O(KSA)$.

Proof. RL-ARNE requires $O(K)$ multiplications for the value and average reward updates (lines 9-10 of Algorithm VI.1), and $O(KA)$ multiplications for the gradient and policy updates (lines 11-12). Hence, per iteration computation complexity is $O(KA)$. The memory required for the RL-ARNE is $O(KS)$ for the value updates (line 9), $O(K)$ for the average reward updates (line 10), and $O(KSA)$ for the gradient and policy updates (lines 11-12). Hence, memory complexity is $O(KSA)$. □

8.5.2 RL-ARNE Algorithm

8.5.3 Convergence Proof of the RL-ARNE Algorithm

First rewrite iterations in line 9 and line 10 as Eqn. (8.11) and Eqn. (8.12) to show the convergence of value and average reward payoff iterates in Algorithm 8.

$$v_k^{n+1}(s) = v_k^n(s) + \delta_v^n[F(v_k^n, \rho_k^n)(s) - v_k^n(s) + w_v^n] \tag{8.11}$$

$$\rho_k^{n+1} = \rho_k^n + \delta_\rho^n[G(\rho_k^n) - \rho_k^n + w_\rho^n] \tag{8.12}$$

For brevity we use $\pi(s,d,a) = \pi_D(s,d)\pi_A(s,a)$ and π to denote (π_D, π_A). Let $\mathbf{P}(s'|s,\pi) = \sum_{d\in\mathcal{A}_D(s)} \sum_{a\in\mathcal{A}_A(s)} \pi(s,d,a)\mathbf{P}(s'|s,d,a)$. Two function maps $F(v_k^n)(s)$ and $G(\rho_k^n)$ are defined as

$$F(v_k^n, \rho_k^n)(s) = \sum_{s'\in\mathbf{S}} \mathbf{P}(s'|s,\pi)[r_k(s,d,a,s') - \rho_k^n + v_k^n(s')], \tag{8.13}$$

$$G(\rho_k^n) = \sum_{s'\in\mathbf{S}} \mathbf{P}(s'|s,\pi)\Big[\frac{n\rho_k^n + r_k(s,d,a,s')}{n+1}\Big]. \tag{8.14}$$

The zero mean noise parameters w_v^n and w_ρ^n are defined as

$$w_v^n = r_k(s,d,a,s') - \rho_k^n + v_k^n(s') - F(v_k^n, \rho_k^n)(s), \tag{8.15}$$

$$w_\rho^n = \frac{n\rho_k^n + r_k(s,d,a,s')}{n+1} - G(\rho_k^n). \tag{8.16}$$

Let $v_k = [v_k(s)]_{s\in\mathbf{S}}$. Then the ODE associated with the iterates given in Eqn. (8.11) corresponding to all $s \in \mathbf{S}$ and the ODE associated with the iterate in Eqn. (8.12) are as follows.

$$\dot{v}_k = f(v_k, \rho_k) \text{ and } \dot{\rho}_k = g(\rho_k), \tag{8.17}$$

where $f : \mathcal{R}^{|\mathbf{S}|} \to \mathcal{R}^{|\mathbf{S}|}$ is such that $f(v_k, \rho_k) = F(v_k, \rho_k) - v_k$, where $F(v_k, \rho_k) = [F(v_k, \rho_k)(s)]_{s\in\mathbf{S}}$ and $g : \mathcal{R} \to \mathcal{R}$ is defined as $g(\rho_k) = G(\rho_k) - \rho_k$. We note that, in Algorithm 8, value function iterates ($v_k^n(s)$) runs in a relatively faster time scale compared to the average reward iterates (ρ_k^n). As a consequence, $v_k^n(s)$ iterates see ρ_k^n as quasi-static. Hence, for brevity, in the proofs of Lemma 8.5.4, Lemma 8.5.7, and Theorem 8.5.9 we represent $f(v_k, \rho_k)$ and $F(v_k^n, \rho_k^n)(s)$ as $f(v_k)$ and $F(v_k^n)(s)$, respectively.

A set of lemmas that are used to prove the convergence of the iterates in lines 9 and 10 of Algorithm 8 are given below. Lemma 8.5.4 presents a property of the ODEs in Eqn. (8.17).

Lemma 8.5.4. Consider the ODEs $\dot{v}_k = f(v_k, \rho_k)$ and $\dot{\rho}_k = g(\rho_k)$. Then the functions $f(v_k, \rho_k)$ and $g(\rho_k)$ are Lipschitz.

Proof. First we show $f(v_k)$ is Lipschitz. Consider two distinct value vectors v_k and $\bar{v}_k$. Then,

$$\begin{aligned}
\| f(v_k) - f(\bar{v}_k)\|_1 =& \| [F(v_k) - F(\bar{v}_k)] - [v_k - \bar{v}_k]\|_1 \\
\leq& \| F(v_k) - F(\bar{v}_k)\|_1 + \| v_k - \bar{v}_k\|_1 \\
=& \sum_{s\in\mathbf{S}} \left| F(v_k)(s) - F(\bar{v}_k)(s)\right| + \| v_k - \bar{v}_k\|_1. \qquad (8.18)
\end{aligned}$$

$$\begin{aligned}
\sum_{s\in\mathbf{S}} \left| F(v_k)(s) - F(\bar{v}_k)(s)\right| &= \sum_{s\in\mathbf{S}} \left| \sum_{s'\in\mathbf{S}} \mathbf{P}(s'|s,\pi)[v_k(s') - \bar{v}_k(s')]\right| \\
&\leq \sum_{s\in\mathbf{S}}\sum_{s'\in\mathbf{S}} \mathbf{P}(s'|s,\pi)\, |v_k(s') - \bar{v}_k(s')| \\
\leq \sum_{s\in\mathbf{S}}\sum_{s'\in\mathbf{S}} |v_k(s') - \bar{v}_k(s')| &= \sum_{s\in\mathbf{S}} \| v_k - \bar{v}_k\|_1 = |\mathbf{S}|\, \| v_k - \bar{v}_k\|_1.
\end{aligned}$$

The inequalities above follow from the triangle inequality and observing the fact that $\max\{\mathbf{P}(s'|s,\pi)\} = 1$. Then from Eqn. (8.18),

$$\| f(v_k) - f(\bar{v}_k)\|_1 \leq (|\mathbf{S}| + 1)\, \| v_k - \bar{v}_k\|_1.$$

Hence $f(v_k)$ is Lipschitz. Next we prove $g(\rho_k)$ is Lipschitz. Let ρ_k and $\bar{\rho}_k$ be two distinct average payoff values. Then,

$$|g(\rho_k) - g(\bar{\rho}_k)| = \left| \frac{n}{n+1}[\rho_k - \bar{\rho}_k] - [\rho_k - \bar{\rho}_k]\right| = |\rho_k - \bar{\rho}_k|\,.$$

Therefore $g(\rho_k)$ is Lipschitz. □

Lemma 8.5.7 shows the map $F(v_k^n) = [F(v_k^n)(s)]_{s\in\mathbf{S}}$ is a pseudo-contraction w.r.t some weighted sup-norm. The definitions of weighted sup-norm and pseudo-contraction are given below.

Definition 8.5.5. Let $||b||_\epsilon$ denote the weighted sup-norm of a vector $b \in \mathcal{R}^{m_b}$ w.r.t vector $\epsilon \in \mathcal{R}^{m_b}$. Then, $||b||_\epsilon = \max_{q=1,\ldots,n} \frac{|b(q)|}{\epsilon(q)}$, where $|b(q)|$ represent the absolute value of the q^{th} entry of vector b.

Definition 8.5.6 (Pseudo contraction). Let $c, \bar{c} \in \mathcal{R}^{m_c}$. Then a function $\phi : \mathcal{R}^{m_c} \to \mathcal{R}^{m_c}$ is said to be a pseudo contraction w.r.t the vector $\gamma \in \mathcal{R}^{m_c}$ if and only if,

$$\| \phi(c) - \phi(\bar{c}) \|_\gamma \leq \eta \| c - \bar{c} \|_\gamma, \text{ where } 0 \leq \eta < 1.$$

Lemma 8.5.7. Consider $F(v_k^n, \rho_k^n)(s)$ defined in Eqn. (8.13). Then the function map $F(v_k^n, \rho_k^n) = [F(v_k^n, \rho_k^n)(s)]_{s \in \mathbf{S}}$ is a pseudo-contraction w.r.t some weighted sup-norm.

Proof. Consider two distinct value functions v_k^n and $\bar{v}_k^n$. Then,

$$\begin{aligned}
&\| F(v_k^n)(s) - F(\bar{v}_k^n)(s) \|_1 = \| \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, \pi)[v_k^n(s') - \bar{v}_k^n(s')] \|_1 \\
&= \| \sum_{s' \in \mathbf{S}} \sum_{d \in \mathcal{A}_D(s),\ a \in \mathcal{A}_A(s)} \pi(s, d, a) \mathbf{P}(s'|s, d, a)[v_k^n(s') - \bar{v}_k^n(s')] \|_1 \\
&\leq \sum_{d \in \mathcal{A}_D(s),\ a \in \mathcal{A}_A(s)} \pi(s, d, a) \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, d, a) \| v_k^n(s') - \bar{v}_k^n(s') \|_1 \qquad (8.19)
\end{aligned}$$

Eqn. (8.19) follows from triangle inequality. To find an upper bound for the term $\mathbf{P}(s'|s, d, a)$ in Eqn. (8.19), we construct a Stochastic Shortest Path Problem (SSPP) with the same state space and transition probability structure as in the game, and a player whose action set is given by $\mathcal{A}_D \times \mathcal{A}_A$. Further set the rewards corresponding to all the state transition in SSPP to be -1. Then by Proposition 2.2 in [**?**], the following condition for all $s \in \mathbf{S}$ and $(d, a) \in \mathcal{A}_D(s) \times \mathcal{A}_A(s)$. Eqn. (8.19) follows from triangle inequality. To find an upper bound for the term $\mathbf{P}(s'|s, d, a)$ in Eqn. (8.19), we construct a Stochastic Shortest Path Problem (SSPP) with the same state space and transition probability structure as in the game, and a player whose action set is given by $\mathcal{A}_D \times \mathcal{A}_A$. Further set the rewards corresponding to all the state transition in SSPP to be -1. Then by Proposition 2.2 in [**?**], $\sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, d, a)\epsilon(s') \leq \eta\epsilon(s)$ holds for all $s \in \mathbf{S}$ and $(d, a) \in \mathcal{A}_D(s) \times$

$\mathcal{A}_A(s)$, where $\epsilon \in [0,1]^{|\mathbf{S}|}$ and $0 \leq \eta < 1$. Rewrite Eqn. (8.19) as $|F(v_k^n)(s) - F(\bar{v}_k^n)(s)|$

$$\leq \sum_{d\in\mathcal{A}_D(s),\ a\in\mathcal{A}_A(s)} \pi(s,d,a) \sum_{s'\in\mathbf{S}} \mathbf{P}(s'|s,d,a)\epsilon(s') \frac{|v_k^n(s') - \bar{v}_k^n(s')|}{\epsilon(s')}$$

$$\leq \sum_{d\in\mathcal{A}_D(s)\ ,a\in\mathcal{A}_A(s)} \pi(s,d,a) \sum_{s'\in\mathbf{S}} \mathbf{P}(s'|s,d,a)\epsilon(s') \parallel v_k^n - \bar{v}_k^n \parallel_\epsilon$$

$$\leq \sum_{d\in\mathcal{A}_D(s),\ a\in\mathcal{A}_A(s)} \pi(s,d,a)\eta\epsilon(s) \parallel v_k^n - \bar{v}_k^n \parallel_\epsilon = \eta\epsilon(s) \parallel v_k^n - \bar{v}_k^n \parallel_\epsilon.$$

$$\frac{|F(v_k^n)(s) - F(\bar{v}_k^n)(s)|}{\epsilon(s)} \leq \eta \parallel v_k^n - \bar{v}_k^n \parallel_\epsilon$$

$$\parallel F(v_k^n) - F(\bar{v}_k^n) \parallel_\epsilon \leq \eta \parallel v_k^n - \bar{v}_k^n \parallel_\epsilon. \qquad \square$$

Below we prove boundedness of Algorithm 8 iterates.

Lemma 8.5.8. Consider the RL-ARNE algorithm presented in Algorithm 8. Then, the iterates $v_k^n(s)$ and ρ_k^n, for $s \in \mathbf{S}$ and $k \in \{D, A\}$, in Eqn.(8.11) and Eqn.(8.12) are bounded.

Proof. Recall Eqns. (8.11) and (8.12). We know $\delta_\rho^n \ll \delta_v^n$. In order to prove the result we first show the errors introduced in the slow iterates $v_k^n(s)$ by the transient errors of the fast iterates ρ_k^n approach to zero as $n \to \infty$. For a positive integer Δ, with slight abuse of notation, we let $v_k^{n+\Delta}(s)$ to denote the value function at the iterate n computed by replacing ρ_k^n at Eqn.(8.11) with $\rho_k^{n+\Delta}$. As a consequence, an error

$$Err(v;\rho) = v_k^{n+\Delta}(s) - v_k^n(s) = \delta_v^n(\rho_k^n - \rho_k^{n+\Delta}), \tag{8.20}$$

is introduced in $v_k^n(s)$ iterates, where the term $\rho_k^n - \rho_k^{n+\Delta}$ captures the transient errors of the fast iterate ρ_k^n. It has been shown that $\rho_k^n - \rho_k^{n+\Delta} = O(\delta_\rho^n)$ in [**?**, **?**]. Then, from Eqn. (8.20) we get $Err(v;\rho) = O(\delta_v^n\delta_\rho^n)$. This proves that $Err(v;\rho) \to 0$ when $n \to \infty$ as $\delta_v^n, \delta_\rho^n \ll 1$ and $\delta_\rho^n \to 0$ at a faster rate compared to δ_v^n due to $\delta_\rho^n \ll \delta_v^n$, when $n \to \infty$. Similarly, since $\delta_\pi^n \ll \delta_v^n$ and $\delta_\pi^n \ll \delta_\rho^n$, we can show $Err(v;\pi),\ Err(\rho;\pi) \to 0$ as $n \to \infty$. Therefore, the error introduced in the slow iterates due to the transient errors of the fast iterates are asymptotically bounded.

Lemma 8.5.7 proved that $F(v_k^n)$ is a pseudo-contraction w.r.t some weighted sup-norm. By choosing step-size, δ_v^n to satisfy Assumption 8.2.6 and observing that the noise parameter, w_v^n is

zero mean with bounded variance, all the conditions in Theorem 1 in [**?**] hold for the game. Hence, by Theorem 1 in [**?**], the iterates $v_k^n(s)$ in Eqn. (8.11) are bounded for all $s \in \mathbf{S}$.

Finally, we show the boundedness of the ρ_k^n iterates. From Proposition 8.2.3, for a fixed policy pair (π_D, π_A) and $n >> 0$, the average reward payoff values ρ_k^n depend only on the rewards due to the state transitions that occur within the recurrent classes of induced MC. Recall that under Assumption 8.2.2 the induced Markov chain, $\mathbf{P}(\pi_D, \pi_A)$, contains only a single recurrent class. Let $\mathbf{S}_1$ be the set of states in the recurrent class of $\mathbf{P}(\pi_D, \pi_A)$. Then there exists a unique stationary distribution p for $\mathbf{P}(\pi_D, \pi_A)$ restricted to states in $\mathbf{S}_1$. Thus for $n >> 0$ and each $k \in \{D, A\}$,

$$\rho_k^n = \sum_{s \in \mathbf{S}_1} p(s) r_k(s, \pi), \tag{8.21}$$

where $p(s)$ is the probability of being at state $s \in \mathbf{S}_1$ and $r_k(s,\pi) = \sum_{d \in \mathcal{A}_D(s),\ a \in \mathcal{A}_A(s)} \pi(s,d)\pi(s,a) \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s,d,a) r(s,$ is the expected reward at the state $s \in \mathbf{S}_1$ for player $k \in \{D, A\}$. Since $\mathbf{S_1}$ has finite cardinality and the rewards r_ks are finite for the game, ρ_k^n converges to a globally asymptotically stable critical point given in Eqn. (8.21) and ρ_k^n iterates are bounded. □

Theorem 8.5.9 proves the convergence of the iterates $v_k^n(s)$ and ρ_k^n.

Theorem 8.5.9. Consider the RL-ARNE algorithm presented in Algorithm 8. Then the iterates $v_k^n(s)$, for all $s \in \mathbf{S}$, and ρ_k^n for $k \in \{D, A\}$ in Eqn. (8.11) and Eqn. (8.12) converge.

Proof. By Proposition 8.2.8, an SA-based algorithm converges under the conditions (I)-(VI). Lemma 8.5.4 and Lemma 8.5.8 showed that conditions (I) and (II) in Proposition 8.2.8 are satisfied, respectively.

To show that condition (III) is satisfied, we first show that there exists a Lipschitz function $\psi_k(\rho_k)$ that characterizes the critical points of $\dot{v}_k = f(v_k, \rho_k)$ in Eqn. (8.17). Note that v_k is a critical point of $\dot{v}_k = f(v_k, \rho_k)$ if and only if $v_k = F(v_k, \rho_k)$. Let $|\mathbf{S}|$ be the cardinality of the state space associated with the game. Let $\mathbf{1}_{|\mathbf{S}|\times 1}$ and $\mathbf{I}_{|\mathbf{S}|\times|\mathbf{S}|}$ denote all ones vector with length $|\mathbf{S}|$ and $|\mathbf{S}| \times |\mathbf{S}|$ identity matrix, respectively. Then using Eqn. (8.13), we get $v_k = \bar{r} - \rho_k \mathbf{1}_{|\mathbf{S}|\times 1} + \mathbf{P}(\pi_D, \pi_A) v_k$, which can be rewritten

$$\big[\mathbf{I}_{|\mathbf{S}|\times|\mathbf{S}|} - \mathbf{P}(\pi_D, \pi_A)\big] v_k = \bar{r} - \rho_k \mathbf{1}_{|\mathbf{S}|\times 1}, \tag{8.22}$$

where $\bar{r}$ is a length $|\mathbf{S}|$ vector whose entries are given by $\sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s,\pi) r_k(s,d,a,s')$. The set of linear equations defined in Eqn. (8.22) has infinite number of solutions under Assumption 8.2.2 [**?**, 50]. Let $\mathbf{J} = \mathbf{I}_{|\mathbf{S}|\times|\mathbf{S}|} - \mathbf{P}(\pi_D, \pi_A)$. Then for an arbitrary vector ω, we define $\psi_k(\rho_k) = \mathbf{J}^+\big[\bar{r} - \rho_k \mathbf{1}_{|\mathbf{S}|\times 1}\big] + \big[\mathbf{I}_{|\mathbf{S}|\times|\mathbf{S}|} - \mathbf{J}^+\mathbf{J}\big]\omega$, where $\mathbf{J}^+$ denotes the generalized inverse [**?**] of $\mathbf{J}$. Consider two distinct ρ_k and $\bar{\rho}_k$ with a fixed ω.

$$\begin{aligned} \| \psi_k(\rho_k) - \psi_k(\bar{\rho}_k)\|_1 &= \| \mathbf{J}^+(\bar{\rho}_k - \rho_k)\mathbf{1}_{|\mathbf{S}|\times 1}\|_1 \\ \leq \| \mathbf{J}^+\|_1 \, \| (\rho_k - \bar{\rho}_k)\mathbf{1}_{|\mathbf{S}|\times 1}\|_1 &= |\mathbf{S}| \, \| \mathbf{J}^+\|_1 \, \| \rho_k - \bar{\rho}_k\|_1 \end{aligned}$$

Hence, $\psi_k(\rho_k)$ is Lipschitz. Next we show $F(v_k^n)(s)$ is a non-expansive map to prove the convergence of the $v_k^n(s)$ iterates. Consider two distinct v_k^n and $\bar{v}_k^n$. Since $P(s'|s,\pi_D,\pi_A) \leq 1$, from Eqn. (8.19), $\| F(v_k^n)(s) - F(\bar{v}_k^n)(s)\| \leq \| v_k^n(s') - \bar{v}_k^n(s')\|$. Thus $F(v_k^n)(s)$ is a non-expansive map and hence from Theorem 2.2 in [**?**] iterates $v_k^n(s)$, for all $s \in \mathbf{S}$ and $k \in \{D, A\}$, converge to an asymptotically stable critical point given by $\psi_k(\rho_k)$. Hence, condition (III) is verified.

Lemma 8.5.8, showed that ρ_k^n, for $k \in \{D, A\}$, converge to a globally asymptotically stable critical point which implies that condition (IV) is satisfied. From Eqns. (8.15) and (8.16), the noise measures have zero mean. The variance of these noise measures are bounded by the fineness of the rewards in the game and the boundedness of the iterates $v_k^n(s)$ and ρ_k^n. Thus condition (V) is satisfied. Finally, the choice of step-sizes satisfies condition (VI). Therefore the results follows by Proposition 8.2.8. □

Next theorem proves the convergence of gradient estimates.

Theorem 8.5.10. Consider $\Omega_{k,\pi_{-k}}^{s,a_k}$ and $\Delta(\pi)$ given in Eqns. (**??**) and (**??**), respectively. Then gradient estimation iterate, $\epsilon_k^n(s,a_k)$ in line 11 corresponding to any $k \in \{D, A\}$, $s \in \mathbf{S}$, and $a_k \in \mathcal{A}_k(s)$, converge to $-\frac{\partial \Delta(\pi)}{\partial \pi_k(s,a_k)} = -\sum_{\bar{k} \in \{A,D\}} \Omega_{\bar{k},\pi_{-k}}^{s,a_k}$.

Proof. Rewrite gradient estimation in line 11 as follows.

$$\epsilon_k^{n+1}(s,a_k) = \epsilon_k^n(s,a_k) + \delta_\epsilon^n\big[-\sum_{\bar{k} \in \{D,A\}} \Omega_{\bar{k},\pi_{-k}}^{s,a_k} - \epsilon_k^n(s,a_k) + w_\epsilon^n\big], \tag{8.23}$$

where $w_\epsilon^n = \sum_{k\in\{D,A\}} \bar{\Omega}_k^s + \sum_{\bar{k}\in\{D,A\}} \Omega_{\bar{k},\pi_{-k}}^{s,a_k}$, and $\bar{\Omega}_k^s = r_k(s,d,a,s') - \rho_k^n + v_k^n(s') - v_k^n(s)$. Since $\mathbb{E}(w_\epsilon^n) = 0$, the ODE associated with Eqn. (8.23) is given by, $\dot{\epsilon}_k(s,a_k) = -\sum_{\bar{k}\in\{D,A\}} \Omega_{\bar{k},\pi_{-k}}^{s,a_k} - \epsilon_k(s,a_k)$.

We use Proposition 8.2.7 to prove the convergence of gradient estimation iterates, $\epsilon_k^n(s,a_k)$. Step-size δ_ϵ^n is chosen such that condition 1) in Proposition 8.2.7 is satisfied. Validity of condition 2) can be shown as follows.

$$\mathbb{E}\left(\lim_{n\to\infty}\left(\sup_{\bar{n}>n}\left|\sum_{l=n}^{\bar{n}}\delta_\epsilon^l w_\epsilon^l\right|^2\right)\right) \leq 4\lim_{n\to\infty}\sum_{l=n}^{\infty}(\delta_\epsilon^l)^2\mathbb{E}(|w_\epsilon^l|^2) = 0. \tag{8.24}$$

Inequality in Eqn. (8.24) follows by Doob inequality [**?**]. Equality in Eqn. (8.24) follows by choosing δ_ϵ^n to satisfy Assumption 8.2.6 and observing $\mathbb{E}(|w_\epsilon^l|^2) < \infty$ as r_k, v_k^n, and ρ_k^n are bounded in the game. Comparing Eqn. (8.23) with Eqn. (8.4), $\kappa = 0$ in Eqn. (8.23). Therefore, from Proposition 8.2.7, as $n\to\infty$, $\epsilon_k^n(s,a_k) \to -\sum_{\bar{k}\in\{A,D\}} \Omega_{\bar{k},\pi_{-k}}^{s,a_k} = -\frac{\partial\Delta(\pi)}{\partial\pi_k(s,a_k)}$. This completes the proof showing the convergence of gradient estimation iterates $\epsilon_k^n(s,a_k)$. □

Next, we prove the convergence of the policy iterates. In order to do so, we proceed in the following manner.

1. We rewrite the conditions in Prop 8.2.5 that characterize ARNE of the game as a non-linear optimization problem (Problem 8.5.11).

2. Then we show the policies are updated in a valid decent direction, $\sqrt{\pi_k^n(s,a_k)}\left|\Omega_{k,\pi_{-k}}^{s,a_k}\right|\text{sgn}\left(\frac{\partial\Delta(\pi^n)}{\partial\pi_k^n(s,a_k)}\right)$, w.r.t the objective function (TD error), $\Delta(\pi)$, of Problem 8.5.11 (Lemma 8.5.12).

3. Using steps 1) and 2), we characterize the stable and unstable equilibrium points associated with the ODE corresponding to the policy iterates in line 12 (Lemma 8.5.13).

4. Invoking Prop 8.2.7 we prove the convergence of policy iterates to stable equilibrium points found in step 3) (Theorem 8.5.14).

Below we elaborate steps 1)-4). The non-linear program below characterizes an ARNE of the game (step 1).

Problem 8.5.11. The necessary and sufficient conditions given in Proposition 8.2.5 that characterize the ARNE of the game can be reformulated as the following non-linear program using $\Omega^{s,a_k}_{k,\pi_{-k}}$ and $\Delta(\pi)$ introduced in Eqns. (**??**) and (**??**).

$$\min_{v,\rho,\pi} \quad \Delta(\pi) \text{ s.t.} \quad \Omega^{s,a_k}_{k,\pi_{-k}} \geq 0; \sum_{a_k \in \mathcal{A}_k(s)} \pi_k(s,a_k) = 1;\ \pi_k(s,a_k) \geq 0,$$

where $v = (v_D, v_A),\ v_k = [v_k(s)]_{s\in\mathbf{S}},\ \rho = (\rho_D, \rho_A),\ \pi = (\pi_D, \pi_A),\ \pi_k = [\pi_k(s)]_{s\in\mathbf{S}},\ \pi_k(s) = [\pi_k(s,a_k)]_{a_k\in\mathcal{A}_k(s)}$, for $k \in \{D, A\}$.

Lemma 8.5.12 proves policy iterates are updated in a valid descent direction w.r.t the objective function, $\Delta(\pi)$ (step 2).

Lemma 8.5.12. Consider $\Omega^{s,a_k}_{k,\pi_{-k}}$, $\Delta(\pi)$ given in Eqs. (**??**), (**??**), respectively. For any $k \in \{D, A\}$, $s \in \mathbf{S}$, and $a_k \in \mathcal{A}_k(s)$, policy iterate, $\pi^n_k(s,a_k)$, in line 12 of Algorithm 8 is updated in a valid descent direction, $\sqrt{\pi^n_k(s,a_k)}\left|\Omega^{s,a_k}_{k,\pi_{-k}}\right|\text{sgn}\left(\frac{\partial\Delta(\pi^n)}{\partial\pi^n_k(s,a_k)}\right)$, of $\Delta(\pi)$ when $\Omega^{s,a_k}_{k,\pi_{-k}} \geq 0$ and $\Delta(\pi) > 0$.

Proof. First we rewrite policy iteration in line 12 as follows.

$$\begin{aligned}\pi^{n+1}_k(s,a_k) = \Gamma(\pi^n_k(s,a_k) - \delta^n_\pi \Bigg(&\sqrt{\pi^n_k(s,a_k)}\left|\Omega^{s,a_k}_{k,\pi_{-k}}\right| \\ &\text{sgn}\left(\frac{\partial\Delta(\pi^n)}{\partial\pi^n_k(s,a_k)}\right) + w^n_\pi\Bigg)),\end{aligned} \tag{8.25}$$

where $w^n_\pi = \sqrt{\pi^n_k(s,a_k)}\left[\left|\bar{\Omega}^s_k\right| - \left|\Omega^{s,a_k}_{k,\pi_{-k}}\right|\right]\text{sgn}\left(\frac{\partial\Delta(\pi^n)}{\partial\pi^n_k(s,a_k)}\right)$, and $\bar{\Omega}^s_k = r_k(s,d,a,s') - \rho^n_k + v^n_k(s') - v^n_k(s)$. Policy iterate updates in the slowest time scale when compared to the other iterates. Thus, all the terms except $\Omega^{s,a_k}_{k,\pi_{-k}}$ in Eqn. (8.25) use the converged values of v_k, ρ_k, and $\frac{\partial\Delta(\pi^n)}{\partial\pi^n_k(s,a_k)}$ w.r.t policy $\pi^n = (\pi^n_D, \pi^n_A)$.

Consider a policy π^{n+1}_k whose entries are same as π^n_k except the entry $\pi^{n+1}_k(s,a_k)$ which is chosen as in Eqn. (8.25), for small $0 < \delta^n_\pi << 1$. Let $\bar{\pi} = (\pi^{n+1}_k, \pi^n_{-k})$ and $\hat{\pi} = (\pi^n_k, \pi^n_{-k})$. Also

note that $\mathbb{E}(w_\pi^n) = 0$. Thus ignoring the term w_π^n and using Taylor series expansion yields,

$$\begin{aligned}\Delta(\bar{\pi}) =&\Delta(\hat{\pi}) + \delta_\pi^n \left(-\sqrt{\pi_k^n(s,a_k)} \left| \Omega_{k,\pi_{-k}}^{s,a_k} \right| \right. \\ & \left. \text{sgn}\left(\frac{\partial \Delta(\hat{\pi})}{\partial \pi_k^n(s,a_k)} \right) \frac{\partial \Delta(\hat{\pi})}{\partial \pi_k^n(s,a_k)} \right) + o(\delta_\pi^n)\end{aligned}$$

$$\Delta(\bar{\pi}) = \Delta(\hat{\pi}) + \delta_\pi^n \left(-\sqrt{\pi_k^n(s,a_k)} \left| \Omega_{k,\pi_{-k}}^{s,a_k} \right| \left| \frac{\partial \Delta(\hat{\pi})}{\partial \pi_k^n(s,a_k)} \right| \right),$$

where $o(\delta_\pi^n)$ represents the higher order terms corresponding to δ_π^n. We ignore $o(\delta_\pi^n)$ in the second equality above since the choice of δ_π^n is small. Notice that the term $\delta_\pi^n \left(-\sqrt{\pi_k^n(s,a_k)} \left| \Omega_{k,\pi_{-k}}^{s,a_k} \right| \left| \frac{\partial \Delta(\hat{\pi})}{\partial \pi_k^n(s,a_k)} \right| \right)$ is negative. Since $\Delta(\pi) > 0$ for any π, we get $\Delta(\bar{\pi}) < \Delta(\hat{\pi})$. This proves policies are updated in a valid descent direction. □

Notice that the ODE associated with Eqn. (8.25) is,

$$\dot{\pi}_k(s,a_k) = \bar{\Gamma} \left(-\sqrt{\pi_k(s,a_k)} \left| \Omega_{k,\pi_{-k}}^{s,a_k} \right| \text{sgn}\left(\frac{\partial \Delta(\pi)}{\partial \pi_k(s,a_k)} \right) \right), \tag{8.26}$$

where $\bar{\Gamma}$ is the continuous version of the projection operator Γ which is defined analogous to the continuous projection operator in Eqn. (8.5). Let Π denotes the set of limit points associated with the system of ODEs in Eqn. (8.26). Let the feasible set of Problem 8.5.11 be

$$H = \{\pi \in L | \Omega_{k,\pi_{-k}}^{s,a_k} \geq 0, \text{ for all } a_k \in \mathcal{A}_k(s),\ s \in \mathbf{S},\ k \in \{D, A\}\}, \tag{8.27}$$

where the set $L = \{\pi | \sum_{a_k \in \mathcal{A}_k(s)} \pi_k(s,a_k) = 1, \pi_k(s,a_k) \geq 0, \text{ for all } a_k \in \mathcal{A}_k(s),\ s \in \mathbf{S}\}$. The set Π can be partitioned using the set H as $\Pi = \Pi_1 \cup \Pi_2$, where $\Pi_1 = \Pi \cap H$ and $\Pi_2 = \Pi \setminus \Pi_1$. Using these notations and steps 1) and 2), we characterize the stable and unstable equilibrium points of the system of ODEs in Eqn. (8.26) in Lemma VI.11 (step 3).

Lemma 8.5.13. The following statements are true for the set of equilibrium policies $\pi^\star$ of ODE in Eqn. (8.26).

1. All $\pi^\star \in \Pi_1$ form a set of stable equilibrium points.

2. All $\pi^\star \in \Pi_2$ form a set of unstable equilibrium points.

Proof. First we show statement 1) holds. Since the set Π_1 is in the feasible set H of Problem 8.5.11 defined in Eqn. (8.27), for any $\pi^\star \in \Pi_1$, there exists some $a_k \in \mathcal{A}_k(s),\ s \in \mathbf{S}$ that satisfy $\Omega^{s,a_k}_{k,\pi_{-k}} \geq 0$. Let $B_\zeta(\pi^\star) = \{\pi \in L | \, \|\pi - \pi^\star\| < \zeta\}$. Then, for any $\pi \in B_\zeta(\pi^\star) \setminus \Pi_1$, there exists a $\zeta > 0$ such that $\Omega^{s,a_k}_{k,\pi_{-k}} > 0$ which yields $\frac{\partial \Delta(\pi)}{\partial \pi_k(s,a_k)} > 0$. This implies sgn $\left(\frac{\partial \Delta(\pi)}{\partial \pi_k(s,a_k)}\right) > 0$.

Hence, $\bar{\Gamma}\left(-\sqrt{\pi_k(s,a_k)}\left|\Omega^{s,a_k}_{k,\pi_{-k}}\right|\text{sgn}\left(\frac{\partial \Delta(\pi)}{\partial \pi_k(s,a_k)}\right)\right) < 0$ for any $\pi \in B_\zeta(\pi^\star) \setminus \Pi_1$. This implies that $\pi_k(s,a_k)$ will decrease when moving away from $\pi^\star \in \Pi_1$. This proves $\pi^\star \in \Pi_1$ is an stable equilibrium point of the system of ODEs given in Eqn. (8.26).

To show statement 2) is true, we first note that for any $\pi^\star \in \Pi_2$, there exists some $a_k \in \mathcal{A}_k(s),\ s \in \mathbf{S}$ such that $\Omega^{s,a_k}_{k,\pi_{-k}} < 0$. Then, for any $\pi \in B_\zeta(\pi^\star) \setminus \Pi_2$, there exists a $\zeta > 0$ such that $\Omega^{s,a_k}_{k,\pi_{-k}} < 0$ which yields $\frac{\partial \Delta(\pi)}{\partial \pi_k(s,a_k)} < 0$. This implies sgn $\left(\frac{\partial \Delta(\pi)}{\partial \pi_k(s,a_k)}\right) < 0$. Therefore, $\bar{\Gamma}\left(-\sqrt{\pi_k(s,a_k)}\left|\Omega^{s,a_k}_{k,\pi_{-k}}\right|\text{sgn}\left(\frac{\partial \Delta(\pi)}{\partial \pi_k(s,a_k)}\right)\right) > 0$ for any $\pi \in B_\zeta(\pi^\star) \setminus \Pi_2$. This implies that $\pi_k(s,a_k)$ will increase when moving away from $\pi^\star \in \Pi_2$. This proves $\pi^\star \in \Pi_2$ is an unstable equilibrium point of the system of ODEs in Eqn. (8.26) and completes the proof. □

Theorem 8.5.14 gives the convergence of the policy iterates to the set of stable equilibrium points in step 3) (step 4).

Theorem 8.5.14. The policy iterates $\pi^n_k(s,a_k)$ for all $a_k \in \mathcal{A}_k(s)$, $s \in \mathbf{S}$, and $k \in \{D,\ A\}$ in Algorithm 8 converge to a stable equilibrium point $\pi^\star = (\pi^\star_D, \pi^\star_A) \in \Pi_1$.

Proof. Recall $w^n_\pi = \sqrt{\pi^n_k(s,a_k)}\left[\left|\bar{\Omega}^s_k\right| - \left|\Omega^{s,a_k}_{k,\pi_{-k}}\right|\right]$ sgn $\left(\frac{\partial \Delta(\pi^n)}{\partial \pi^n_k(s,a_k)}\right)$. We invoke Proposition 8.2.7 to prove the convergence of policy iterates, $\pi^n_k(s,a_k)$. Step-size δ^n_π is chosen such that condition 1) in Proposition 8.2.7 is satisfied. Validity of condition 2) can be shown as follows.

$$\mathbb{E}\left(\lim_{n\to\infty}\left(\sup_{\bar{n}>n}\left|\sum_{l=n}^{\bar{n}}\delta^l_\pi w^l_\pi\right|^2\right)\right) \leq 4\lim_{n\to\infty}\sum_{l=n}^{\infty}(\delta^l_\pi)^2\mathbb{E}(|w^l_\pi|^2) = 0. \tag{8.28}$$

In Eqn. (8.28), inequality follows by Doob inequality [**?**] and equality follows by choosing δ^n_π to satisfy Assumption 8.2.6 and observing $\mathbb{E}(|w^l_\pi|^2) < \infty$ as r_k, v^n_k, and ρ^n_k are bounded in the game. Comparing Eqs. (8.25) and (8.4), $\kappa = 0$. Therefore, from Proposition 8.2.7, as $n \to \infty$, the policy iterates $\pi^n_k(s,a_k)$ for all $a_k \in \mathcal{A}_k(s)$, $s \in \mathbf{S}$, and $k \in \{D,\ A\}$ converge to a stable equilibrium point $\pi^\star \in \Pi_1$. □

Next theorem proves the convergence of $\pi_k^n(s, a_k)$ given in line 12 of Algorithm 8, to an ARNE in the game.

Theorem 8.5.15. Consider $\Omega_{k,\pi_{-k}}^{s,a_k}$ and $\Delta(\pi)$ given in Eqns. (**??**) and (**??**), respectively. A converged policy $(\pi_D^\star, \pi_A^\star)$ of Algorithm 8 forms an ARNE in the game.

Proof. In the following, we show any converged policy $\pi^\star = (\pi_D^\star, \pi_A^\star)$ returned by RL-ARNE algorithm presented in Algorithm 8 will satisfy conditions (8.9a)-(8.9c) in Corollary 8.2.5 and thus $\pi^\star$ forms an ARNE in the game.

Recall from Theorem 8.5.14, the policy iterates $\pi_k^n(s, a_k)$ for all $a_k \in \mathcal{A}_k(s)$, $s \in \mathbf{S}$, and $k \in \{D,\ A\}$ converge to a stable equilibrium point $\pi^\star \in \Pi_1$. Also, recall Π denotes the set of limit points associated with the system of ODEs in Eqn. (8.26) and $L = \{\pi | \sum_{a_k \in \mathcal{A}_k(s)} \pi_k(s, a_k) = 1, \pi_k(s, a_k) \geq 0, \text{ for all } a_k \in \mathcal{A}_k(s),\ s \in \mathbf{S}\}$. Then, from the definition of the set Π_1, any converged $\pi^\star$ will satisfy conditions (8.9a) and (8.9c), since $\pi^\star \in \Pi_1 = \Pi \cap H$ yields $\pi^\star \in H$, where $H = \{\pi \in L | \Omega_{k,\pi_{-k}}^{s,a_k} \geq 0, \text{ for all } a_k \in \mathcal{A}_k(s),\ s \in \mathbf{S},\ k \in \{D, A\}\}$.

Then it suffices to show any $\pi^\star \in \Pi_1$ will yield $\sqrt{\pi_k(s, a_k)}\Omega_{k,\pi_{-k}}^{s,a_k} = 0$ since this proves condition (8.9c) in Corollary 8.2.5. We show this by contradiction arguments.

Note that $\bar{\Gamma}\left(-\sqrt{\pi_k(s, a_k)}\left|\Omega_{k,\pi_{-k}}^{s,a_k}\right|\text{sgn}\left(\frac{\partial\Delta(\pi)}{\partial\pi_k(s,a_k)}\right)\right) = 0$ as $\pi^\star$ forms a set of equilibrium polices associated with the system of ODEs in Eqn. (8.26). Suppose there exists a policy $0 < \pi_k(s, a_k) \leq 1$ for some $\bar{a}_k \in \mathcal{A}_k(s)$, $s \in \mathbf{S}$, and $k \in \{D,\ A\}$ such that $\sqrt{\pi_k(s, \bar{a}_k)}\Omega_{k,\pi_{-k}}^{s,a_k} \neq 0$. Consider the following two cases.

Case I: $\pi_k(s, \bar{a}_k) = 1$ **and** $\Omega_{k,\pi_{-k}}^{s,\bar{a}_k} \neq 0$.

Recall $F(v_k, \rho_k) = [F(v_k, \rho_k)(s)]_{s\in\mathbf{S}}$ and $F(v_k, \rho_k)(s) = \sum_{s'\in\mathbf{S}} \mathbf{P}(s'|s, \pi)[r_k(s, d, a, s') - \rho_k^n + v_k(s')]$. Then under Case I $\sum_{a_k\in\mathcal{A}_k(s)} \pi_k(s, a_k)\Omega_{k,\pi_{-k}}^{s,a_k} = \pi_k(s, \bar{a}_k)\Omega_{k,\pi_{-k}}^{s,\bar{a}_k} = 0$, where the first equality is due to $\pi_k(s, \bar{a}_k) = 0$ and the second equality is due to the convergence of the value iterates to their true values (i.e., as $n \to \infty$, $v_k \to F(v_k, \rho_k)$) which is proved in Theorem 8.5.9.

Further, as $\pi_k(s, \bar{a}_k) = 1$ this yields $\Omega_{k,\pi_{-k}}^{s,\bar{a}_k} = 0$, which contradicts the condition $\Omega_{k,\pi_{-k}}^{s,\bar{a}_k} \neq 0$ in Case I.

Case II: $0 < \pi_k(s, \bar{a}_k) < 1$ **and** $\Omega_{k,\pi_{-k}}^{s,\bar{a}_k} \neq 0$.

Under this case we get

$$\bar{\Gamma}\left(-\sqrt{\pi_k(s,a_k)}\left|\Omega^{s,a_k}_{k,\pi_{-k}}\right|\text{sgn}\left(\frac{\partial\Delta(\pi)}{\partial\pi_k(s,a_k)}\right)\right)$$
$$=-\sqrt{\pi_k(s,a_k)}\left|\Omega^{s,a_k}_{k,\pi_{-k}}\right|\text{sgn}\left(\frac{\partial\Delta(\pi)}{\partial\pi_k(s,a_k)}\right)\neq 0,$$

due to conditions in given in the Case II and assuming[3] $\text{sgn}(\cdot)\neq 0$. However this contradicts with our initial observation of $\bar{\Gamma}\left(-\sqrt{\pi_k(s,a_k)}\left|\Omega^{s,a_k}_{k,\pi_{-k}}\right|\text{sgn}\left(\frac{\partial\Delta(\pi)}{\partial\pi_k(s,a_k)}\right)\right)=0$.

Therefore, by contradiction, there does not exist any policy $0<\pi_k(s,a_k)\leq 1$ for some $\bar{a}_k\in\mathcal{A}_k(s)$, $s\in\mathbf{S}$, and $k\in\{D,\ A\}$ such that $\sqrt{\pi_k(s,\bar{a}_k)}\Omega^{s,a_k}_{k,\pi_{-k}}\neq 0$. This proves condition (8.9c) in Corollary 8.2.5 holds. Since conditions (8.9a)-(8.9c) in Corollary 8.2.5 hold, a converged policy $(\pi^\star_D,\pi^\star_A)$ of RL-ARNE algorithm presented in Algorithm 8 forms an ARNE in the game. □

Remark 8.5.16. Note that RL-ARNE algorithm presented in Algorithm VI and the associated convergence proofs given in Section VI.B extend to K-player, non-zero sum, average reward unichain stochastic games. Unichain property is a mild regularity assumption compared to other regularity conditions such as ergodicity or irreducibility [**?**].

8.6 Simulation Study

In this section we test Algorithm 8 on two real-world attack datasets corresponding to a ransomware attack and a nation-sate attack. We first provide a brief explanation on the datasets and the extraction of the IFG from the respective datasets. Then we explain the choice of parameters used in our simulations and present the simulation results.

8.6.1 Attack Datasets

The datasets consist of system logs with both benign and malicious information flows recorded in a *Linux* computer threatened by the following two attacks.

[3]This can be achieved by repeating an action in Algorithm 8 when $\text{sgn}(\cdot)=0$. A similar approach has been proposed in the algorithm that computes an NE of discounted stochastic games in [104].

Ransomware Attack Dataset

The goal of the ransomware attack is to open and read all the files in the $./home$ directory of the victim computer and delete all of these files after writing them into an encrypted file named $ransomware.encrypted$. System logs were recorded by RAIN system [64] and the targets of the ransomware attack (destinations) were annotated in the system logs. Two network sockets that indicate series of communications with external IP addresses in the recorded system logs were identified as the entry points of the attack. The attack consists of three stages, where stage 1 correspond to privilege escalation, stage 2 relate to lateral movement of the attack, and stage 3 represent achieving the goal of encrypting and deleting $./home$ directory. Immediate conversion of the system logs resulted in an information flow graph, $\bar{\mathcal{G}}$, with 173 nodes and 2426 edges.

Nation-State Attack Dataset

Nation state attack is a three day state-of-the-art APT attack implemented by US DARPA red-team during the evaluation of RAIN system log recording system [64]. The goal of the attack is to steal sensitive proprietary and personal information from a Linux server. In our experiments we used the system logs recorded during the day 1 of the attack. The attack consists of four key stages: initial compromise, internal reconnaissance, foothold establishment, and data exfiltration. Adversary entered the system through spear phishing attack to lead the victim to a website that was hosting ads from a malicious web server. Then the adversary exploit a vulnerability in the Firefox browser in the first stage of the attack (initial compromise). Next the adversary fingerprints the compromised system to detect running processes and network information during stage two (internal reconnaissance). In stage three (foothold establishment) adversary writes a malicious program to disk. Lastly in fourth and final stage (data exfiltration) adversary establishes a backdoor to continuously exfiltrates the companies sensitive data from the victim computer system.

8.6.2 Construction of IFG from Dataset

The attack related subgraphs were extracted from $\bar{\mathcal{G}}$ using the following graph pruning steps.

1. For each pair of nodes in $\bar{\mathcal{G}}$ (e.g., process and file, process and process), collapse any existing multiple edges between two nodes to a single directed edge representing the direction of the collapsed edges.

2. Extract all the nodes in $\bar{\mathcal{G}}$ that have at least one information flow path from an entry point of the attack to a destination of stage one of the attack.

3. Extract all the nodes in $\bar{\mathcal{G}}$ that have at least one information flow path from $\mathcal{D}_j$ to $D_{j'}$, for all $j, j' \in \{1, \ldots, M\}$ such that $j \neq j'$.

4. From $\bar{\mathcal{G}}$, extract the subgraph corresponding to the entry points, destinations, and the set of nodes extracted in steps $2)$ and $3)$.

5. Combine all the file-related nodes in the extracted subgraph corresponding to a directory into a single node $(e.g., ./home, ./user)$ in the victim's computer.

6. If the resulting subgraph contains any cycles use *node versioning* techniques [**?**] to remove cycles while preserving the information flow dependencies in the graph.

The resulting graph is called as the pruned IFG. The pruned IFG corresponding to the ransomware attack contains 18 nodes and 29 edges (Figure 8.1).

8.6.3 Parameters

We use the following learning rates in our simulations: $\delta_v^n = \delta_\epsilon^n = 0.5$ if $n < 7000$ and $\delta_v^n = \delta_\epsilon^n = \frac{1.6}{\kappa(s,n)}$, otherwise. $\delta_\rho = \delta_\pi^n = 1$, if $n < 7000$ and $\delta_\rho = \frac{1}{1+\tau(n)\log(\tau(n))}$, $\delta_\pi^n = \frac{1}{\tau(n)}$, otherwise. Note that the learning rates remain constant until iteration 7000 and then start decaying. We observed that setting learning rates in this fashion helps the finite time convergence of the algorithm. Here, the term $\kappa(s, n)$ in δ_v^n and δ_ϵ^n denotes the total number of times a state $s \in \mathbf{S}$ is visited from 7000^{th} iteration onwards in Algorithm 8. Hence, in our simulations, the learning rates δ_v^n of $v_k^n(s)$ iterates

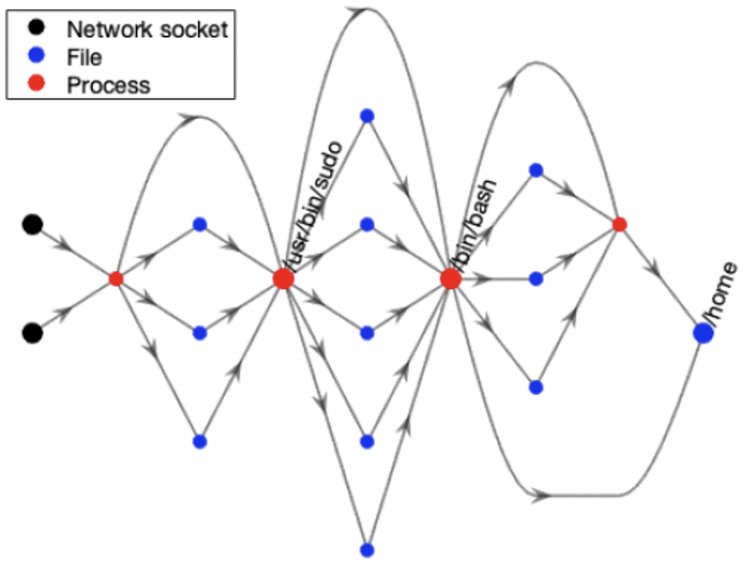


Figure 8.1: IFG of ransomware attack. Nodes of the graph are color coded to illustrate their respective types (network socket, file, and process). Two network sockets are identified as the entry points of the ransomware attack. Destinations of the attack ($/usr/bin/sudo, /bin/bash, /home$) are labeled.

and δ_ϵ^n of the $\epsilon_k^{n+1}(s, a_k)$ iterates depend on the iteration n and the state visited at iteration n. The term $\tau(n) = n - 6999$.

The cost, reward, and penalty parameters used in the simulations of two datasets are described below.

Ransomware Attack

Cost parameters: for all $s_i^j \in \mathbf{S}$ such that $u_i \notin \mathcal{D}_j$, $\mathcal{C}_D(s_i^j) = -0.3$ for $j = 1$, $\mathcal{C}_D(s_i^j) = -0.2$ for $j = 2$, and $\mathcal{C}_D(s_i^j) = -0.1$ for $j = 3$. For all other states, $s \in \mathbf{S}$, $\mathcal{C}_D(s) = 0$. Rewards: $\alpha_D^1 = 0.4$, $\alpha_D^2 = 0.6$, $\alpha_D^3 = 0.8$, $\beta_A^1 = 0.5$, $\beta_A^2 = 0.7$, $\beta_A^3 = 0.9$, $\sigma_D^1 = 0.3$, $\sigma_D^2 = 0.5$, and $\sigma_D^3 = 0.7$. Penalties: $\alpha_A^1 = -0.5$, $\alpha_A^2 = -0.7$, $\alpha_A^3 = -0.9$, $\beta_D^1 = -0.5$, $\beta_D^2 = -0.7$, $\beta_D^3 = -0.9$, $\sigma_A^1 = -0.3$, $\sigma_A^2 = -0.5$, and $\sigma_A^3 = -0.7$.

Nation-State Attack

Cost parameters: for all $s_i^j \in \mathbf{S}$ such that $u_i \notin \mathcal{D}_j$, $\mathcal{C}_D(s_i^j) = -0.4$ for $j = 1$, $\mathcal{C}_D(s_i^j) = -0.3$ for $j = 2$, $\mathcal{C}_D(s_i^j) = -0.2$ for $j = 3$ and $\mathcal{C}_D(s_i^j) = -0.1$ for $j = 4$. For all other states, $s \in \mathbf{S}$, $\mathcal{C}_D(s) = 0$. Rewards: $\alpha_D^1 = 0.15$, $\alpha_D^2 = 0.35$, $\alpha_D^3 = 0.55$, $\alpha_D^4 = 0.75$, $\beta_A^1 = 0.3$, $\beta_A^2 = 0.5$, $\beta_A^3 = 0.7$, $\beta_A^4 = 0.9$, $\sigma_D^1 = 0.3$, $\sigma_D^2 = 0.5$, $\sigma_D^3 = 0.7$, and $\sigma_D^4 = 0.9$. Penalties: $\alpha_A^1 = -0.15$, $\alpha_A^2 = -0.35$, $\alpha_A^3 = -0.55$, $\alpha_A^4 = -0.75$, $\beta_D^1 = -0.15$, $\beta_D^2 = -0.35$, $\beta_D^3 = -0.55$, $\beta_D^4 = -0.75$, $\sigma_A^1 = -0.3$, $\sigma_A^2 = -0.5$, $\sigma_A^3 = -0.7$, and $\sigma_A^4 = -0.9$.

8.6.4 Convergence of RL-ARNE algorithm to an ARNE of DIFT-APT game.

Conditions (8.9a) and (8.9b) in Corollary 8.4.4 are used to validate the convergence of Algorithm 8 to an ARNE of the DIFT-APT game. Let $\phi_T(\pi, \rho, v) = \phi_D(\pi, \rho_D, v_D) + \phi_A(\pi, \rho_A, v_A)$, where $\pi = (\pi_D, \pi_A)$, $\rho = (\rho_D, \rho_A)$, $v = (v_D, v_A)$. Here, $\phi_k(\pi, \rho_k, v_k)$, for $k \in \{D, A\}$, is given by

$$\begin{aligned}\phi_k(\pi, \rho_k, v_k) = &\sum_{s \in \mathbf{S}} \sum_{a_k \in \mathcal{A}_k(s)} \Big(\rho_k + v_k(s) - r_k(s, a_k, \pi_{-k}) \\ &- \sum_{s' \in \mathbf{S}} \mathbf{P}(s'|s, a_k, \pi_{-k}) v_k(s')\Big) \pi_k(s, a_k) = 0\end{aligned} \tag{8.29}$$

We refer to $\phi_T(\pi, \rho, v)$, $\phi_D(\pi, \rho_D, v_D)$, and $\phi_A(\pi, \rho_A, v_A)$ as the total Temporal Difference error (TD error) , DIFT's TD error, and APT's TD error, respectively. Then conditions (8.9a) and (8.9b) in Corollary 8.4.4 together imply that a policy pair forms an ARNE if and only if $\phi_D(\pi, \rho_D, v_D) = \phi_A(\pi, \rho_A, v_A) = 0$. Consequently, at ARNE $\phi_T(\pi, \rho, v) = 0$.

8.6.5 Simulation Results

In following we present the simultation results corresponding to ransomware and nation-state datasets.

Ransomware Attack

Figure 8.2a plots ϕ_T, ϕ_D, ϕ_A corresponding to the policies given by Algorithm 8 at iterations $n = 1, 500, \ldots, 1 \times 10^6$. The plots show that ϕ_T, ϕ_D and ϕ_A converge very close to 0 as n

increases. Figure 8.2b plots the average reward values of DIFT and APT in Algorithm 8 at $n = 1, 500, \ldots, 1 \times 10^6$. Figure 8.2b shows that ρ_D^n, ρ_A^n converge as the iteration count n increases.

Figure 8.2c compares the average rewards of the players corresponding to the converged policies in Algorithm 8 (ARNE policy) against the average reward values of the players corresponding to two other policies of DIFT, i) uniform policy and ii) cut policy. Note that, in i) DIFT chooses an action at every state under a uniform distribution. Where as in ii) DIFT performs security analysis at a destination related state, $s_i^j : u_i \in \mathcal{D}_j$, with probability one whenever the state of the game is an in-neighbor of that destination related state. APT's policy in both uniform policy and cut policy cases are maintained to be as same as in the case of ARNE policy case. The results show that DIFT achieves a higher average reward under ARNE policy when compared to the uniform and cut policies. Further, results also suggest that APT gets a lower reward under the DIFT's ARNE policy when compared to the uniform and cut policies.

Nation-State Attack

Figure 8.3a plots ϕ_T, ϕ_D, ϕ_A related to the policies output by Algorithm 8 at iterations $n = 1, 500, \ldots, 1 \times 10^6$. The results show that ϕ_T, ϕ_D and ϕ_A converge very close to 0 as n increases. Figure 8.3b plots the average reward values of DIFT and APT against the iterations of Algorithm 8, $n = 1, 500, \ldots, 1 \times 10^6$. Figure 8.3b shows that ρ_D^n, ρ_A^n converge as the iteration count n increases. Figure 8.3c compares the average rewards of the players corresponding to ARNE policy against uniform and cut policies. The results show that using ARNE policy output by Algorithm 8, DIFT achieve higher average reward compared to the uniform or cut policies.

8.7 Summary

In this paper we studied the problem of resource efficient and effective detection of Advanced Persistent Threats (APTs) using Dynamic Information Flow Tracking (DIFT) detection mechanism. Making use of the system log data information and postmortem/offline analysis of the data, we constructed the information flow graph. We then modeled the strategic interactions between DIFT

and APT as a nonzero-sum, average reward stochastic game. Additionally, the game has incomplete and imperfect information structure due to unknown transition probabilities and simultaneous actions of the players, respectively.

We showed that the state space of the DIFT-APT game inherently has an unichain structure and proved the existence of an Nash Equibirum in DIFT-APT game. Then we propose RL-ARNE, an actor-critic algorithm that is guaranteed to converge to ARNE of the average reward game under unichain structure, to learn an Average Reward Nash Equilibrium (ARNE) of the DIFT-APT game. The proposed algorithm is an actor-critic algorithm. We proved the convergence of RL-ARNE algorithm to an ARNE of the DIFT-APT game.

We evaluated our game model and algorithm on real-world ransomware and nation-state attack datasets collected using RAIN system log recording system. Our simulation results validated the convergence of the proposed algorithm on the attack datasets and also showed the effectiveness of the RL-ARNE over other heuristic policies.

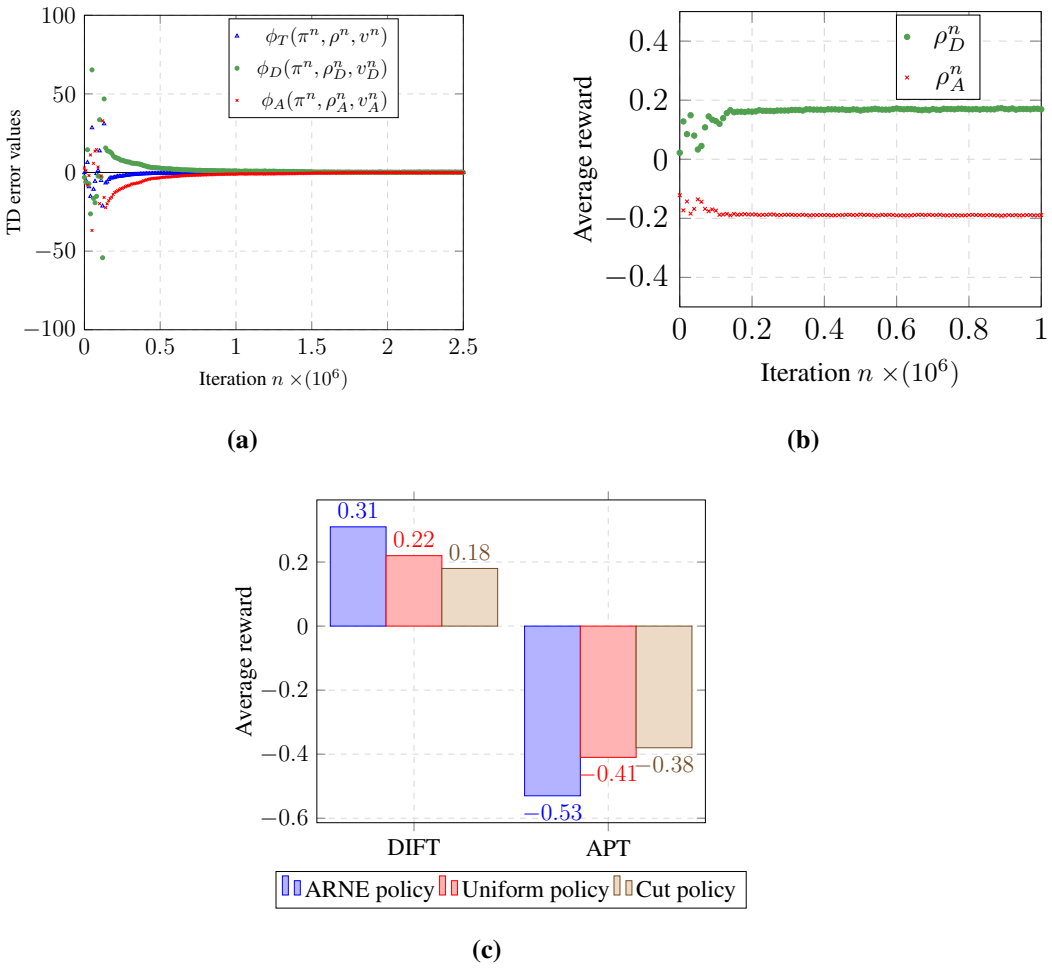


Figure 8.2: Ransomware Attack. (a) Plots of total TD error, $\phi_T(\pi^n, \rho^n, v^n)$, DIFT's TD error $\phi_D(\pi^n, \rho_D^n, v_D^n)$, and APT's TD error $\phi_A(\pi^n, \rho_A^n, v_A^n)$ evaluated at iterations $n = 1, 500, 1000, \ldots, 1 \times 10^6$ of Algorithm 8 for ransomware attack. Values of TD errors at the n^{th} iteration depend on the policies π^n, average rewards ρ^n, and value functions v^n of DIFT and APT at the n^{th} iteration. $\phi_D(\pi^n, \rho_D^n, v_D^n)$ and $\phi_A(\pi^n, \rho_A^n, v_A^n)$ are defined in Eqn. (8.29) and $\phi_T(\pi^n, \rho^n, v^n) = \phi_D(\pi^n, \rho_D^n, v_D^n) + \phi_A(\pi^n, \rho_A^n, v_A^n)$. (b) Plots of the average rewards of DIFT (ρ_D^n) and APT (ρ_A^n) at iteration $n \in \{1, 500, \ldots 1 \times 10^6\}$ of Algorithm 8. Average rewards at the n^{th} iteration depend on the policies π^n of DIFT, APT. (c) Comparison of the average rewards of DIFT and APT obtained by the converged policies in Algorithm 8 (ARNE policy) against the average rewards of the players obtained by two other policies of DIFT: uniform policy and cut policy. Uniform policy: DIFT chooses an action at every state under a uniform distribution. Cut policy: DIFT performs security analysis at a destination related state, $s_i^j : u_i \in \mathcal{D}_j$, with probability one whenever the state of the game is an in-neighbor of that destination related state.

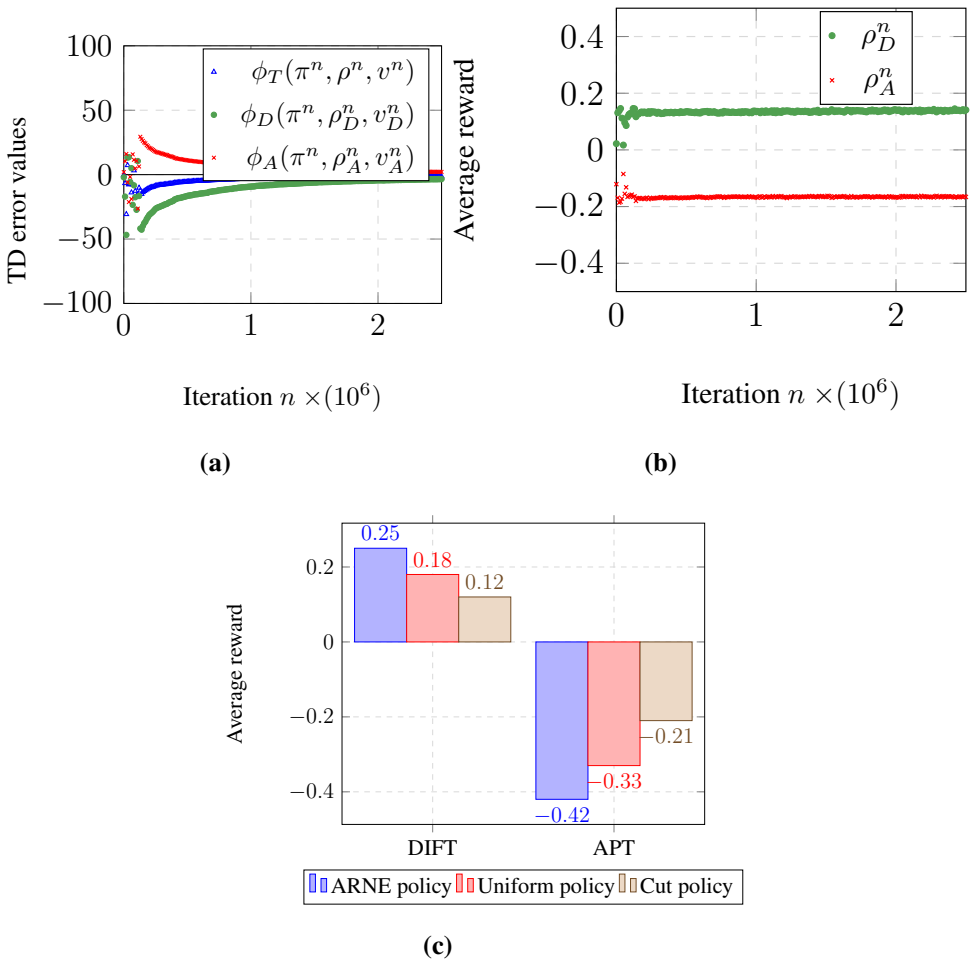


Figure 8.3: Nation-State Attack. (a) Plots of total TD error, $\phi_T(\pi^n, \rho^n, v^n)$, DIFT's TD error $\phi_D(\pi^n, \rho^n_D, v^n_D)$, and APT's TD error $\phi_A(\pi^n, \rho^n_A, v^n_A)$ evaluated at iterations $n = 1, 500, 1000, \ldots, 1 \times 10^6$ of Algorithm 8 for nation-state attack. Values of TD errors at the n^{th} iteration depend on the policies π^n, average rewards ρ^n, and value functions v^n of DIFT and APT at the n^{th} iteration. $\phi_D(\pi^n, \rho^n_D, v^n_D)$ and $\phi_A(\pi^n, \rho^n_A, v^n_A)$ are defined in Eqn. (8.29) and $\phi_T(\pi^n, \rho^n, v^n) = \phi_D(\pi^n, \rho^n_D, v^n_D) + \phi_A(\pi^n, \rho^n_A, v^n_A)$. (b) Plots of the average rewards of DIFT (ρ^n_D) and APT (ρ^n_A) at iteration $n \in \{1, 500, \ldots 1 \times 10^6\}$ of Algorithm 8. Average rewards at the n^{th} iteration depend on the policies π^n of DIFT, APT. (c) Comparison of the average rewards of DIFT and APT obtained by the converged policies in Algorithm 8 (ARNE policy) against the average rewards of the players obtained by two other policies of DIFT: uniform policy and cut policy. Uniform policy: DIFT chooses an action at every state under a uniform distribution. Cut policy: DIFT performs security analysis at a destination related state, $s^j_i : u_i \in \mathcal{D}_j$, with probability one whenever the state of the game is an in-neighbor of that destination related state.

Chapter 9

A NATURAL LANGUAGE PROCESSING APPROACH FOR INSTRUCTION SET ARCHITECTURE IDENTIFICATION IN PROGRAM BINARIES

9.1 Motivation

Modern technological devices such as computers and mobile phones contain firmware and software that consist of thousands of lines of source code [46]. Developers deliberately make the source code unavailable to the public for proprietary and security reasons [59, 107]. Methods to restrict access to the source code include encryption and obfuscation in order to restrict software piracy and mitigate malicious code injection [22, 96]. On the other hand, the source code of malicious software (malware) can also be obfuscated by cyber adversaries to avoid being analyzed by security experts [92]. Consequently, cyber forensics applications such as assessing potential program vulnerabilities [77, 125] and malware detection [33, 56, 138] study the functionality of software by analyzing their binary files. Binary files store the compiled source code as a string of “0"s and “1"s (binaries) interpretable to computer processors. Identifying control paths, data flows, and data types required for assessing content and structure of undisclosed source code often requires converting binaries to assembly language [15, 69, 139]. This is called *disassembly process*.

The disassembly process requires information about the type of processor Instruction Set Architecture (ISA) on which the binaries are expected to run, instruction length, and endianness [15] (Examples in Table 9.1).

Table 9.1 Examples for different disassembly information

Disassembly Info.	Examples
Instruction set architecture	ARM, MIPS, x86, AVR, PowerPC, SPARC, s390
Instruction length	16-bit, 32-bit, 64-bit
Endianness	little-endian, big-endian

The disassembly information is extracted from the file header (e.g., ELF header, PE header) or file name (e.g., .exe for 32-bit x86, 64.exe for 64-bit x86) of binary files. However, these meta-data can be incomplete or missing due to compilation errors or partial downloads. Cyber adversaries have been observed to tamper with these metadata fields in order to remain hidden after carrying out an attack [44, 88].

In the absence of information on ISA type, multi-architecture disassemblers such as Ghidra [3], IDA Pro [4], and Capstone [34], and firmware analysis tools such as Binwalk [110] have been employed to predict the target ISA. Such methods are based on brute forced disassembling of binaries and inspecting assembly codes for known architecture-specific signatures. However, modern computer systems like severs consist of multiple devices such as CPUs, GPUs, and network cards, each of which uses different types of virtual and physical process architectures [58]. The increasing number of different ISA types makes brute forced disassembling-based ISA identification computationally infeasible. The lack of information on target ISA can thwart the disassembly process and affect many cyber forensics applications such as software security assessments [71,83] and cyber threat hunting [27, 105, 141]. Consequently, methods that enable accurate prediction of target ISA solely based on information extracted from (partial) binaries have gained attention recently.

Machine Learning (ML) has been explored as a methodology to predict ISA using features extracted from the object code section of binary files (e.g., byte-histograms and byte signatures)

[21,42,68,85,98,122]. However, accuracy and scalability of ML-based techniques are affected by the following limitations of existing object code features. (i) **Lack of generalizability**: Preselected byte signatures included in feature set for capturing properties of ISA such as endianness may not necessarily be present in some (partial) binaries, (ii) **Noisy data**: The byte patterns that are commonly observed across the binaries of different ISAs (e.g., 0x00[1] byte patterns) are assigned high importance, leading to erroneous predictions of ISAs, and (iii) **Low resolution**: Byte-level granularities might not capture fine-grained bit patterns embedded in the object code of binary files. Enhancing the capabilities of ML-based ISA identification therefore requires an approach to characterize features that will highlight ISA-specific bit patterns that are frequently present in binaries of the corresponding architecture.

We observe that natural language processing (NLP) research provides wide range of models to extract features from text data for applications such as document classification and email filters [40]. We then observe that the object code binaries (i.e., machine language) which define instructions for machines are generated according to a set of well-defined rules similar to the natural language texts that convey messages to human. Based on these two observations, we propose to use models from natural language processing (NLP) to extract features from object code binaries. Specifically, we use the N-gram term frequency-inverse document frequency (TF-IDF), a widely explored technique in NLP research for extracting a set of informative and discriminating text patterns [11, 108]. The binary code feature extraction techniques that we introduce do not require any expert knowledge about the ISAs. We make the following contributions.

- We observe that successive bytes in binaries have co-occurring patterns specific to the ISA. We use byte-level N-gram TF features to extract such patterns.

- We scale byte-level N-gram TF features using their respective IDF values to reduce the effect of noisy data and increase the sensitivity to byte patterns that characterize the ISA.

- We observe that instruction bytes of binaries have fine grained bit patterns specific to the

[1]This paper used the standard prefix notation 0x to indicate that the subsequent number is in the hexadecimal format.

ISA and we use character-level N-gram TF-IDF features of encoded binaries to extract such patterns.

- We use two different datasets with binaries from 12 ISAs and 23 ISAs to show that byte-level $(1, 2, 3)$-gram TF-IDF features yield high accuracy ($\sim 98\%$) compared to the existing byte-histogram and signature-based features ($\sim 91\%$).

- We show that character-level $(1, 2, 3)$-gram TF-IDF features extracted from encoded binaries yield high accuracy with $16\times$ fewer features compared to the number of byte-level $(1, 2, 3)$-gram TF-IDF features.

The remainder of this paper is organized as follows. Section 9.2 provides the preliminaries on binary files and N-gram TF-IDF feature model from NLP. Section 9.3 presents related work. Section 9.4 presents the proposed NLP and encoding-based object code feature selection methods. Sections 9.5 and 9.6 details the experiments and presents the experimental results, respectively. Section 9.7 provides concluding remarks.

9.2 Preliminaries on Binary Files and Binary-to-Text Encoding

In this section we present preliminaries on binary files and some characteristics of binaries that can be leveraged to identify the ISA.

9.2.1 Structures of Binary Files and Instruction Sets

We first detail the relevant components of the binary files studied in this paper. Then we introduce the concept of an ISA and two components of a binary instruction: opcode and operand.

A binary file consists of instructions and resources (e.g., data values, memory addresses, file meta-data such as file size) that are stored as “0"s and “1"s (binaries). Typically binaries are structured as 8-bit (1-byte) terms (e.g., 11010111 01000011). An 8-bit binary term is often represented as a two-digit Hexadecimal (Hex) number for ease of analysis (e.g., 11010111 01000011 in binary =

d7 43 in hex). In this paper we focus on binary files corresponding to executable files, compiled programs, and operating system files that can be processed and executed by a computer to carry out tasks (e.g., computer programs).

File header (e.g., ELF header, PE header) and object code are two important components of binary files required for their analysis. File header provides file meta-data such as file size, instruction length (e.g., 16-bit, 32-bit, 64-bit), details of any object code sections, and instruction set architecture of the processor that runs the binaries. The object code section contains binaries related to set of instructions. In this paper we assume file headers of the binary files are missing and only (partial) object code of binary files are available for analysis.

Instruction Set Architecture (ISA) specifies rules for interpreting instructions in object code binaries to the processor. Examples of popular ISAs include ARM, MIPS, and x86_64. The instruction length of an ISA indicates the number of consecutive bytes that defines one instruction for the processor. Architectures such as ARM and MIPS support fixed length (mostly 32-bit) instructions (Fig. 9.1 and Fig. 9.2) whereas x86_64 supports variable length instructions (Fig. 9.3).

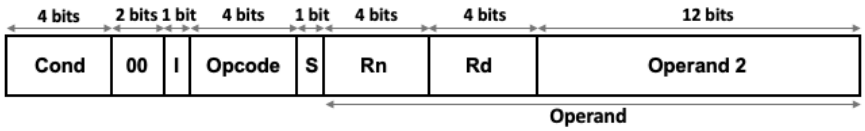


Figure 9.1: 32-bit data processing instruction format of ARM architecture [1]. Cond: Condition field, I: Immediate operand, Opcode: Operation code, S: Set condition codes, Rn: 1st operand register, Rd: Destination register. Typically, the opcode which defines the arithmetic and data operations (e.g., Add, Store) to be carried out by the processor occupies fewer bits in an instruction compared to the operand which specifies the data values and memory/register addresses.

Opcode and operand are two main parts of a binary instruction. The opcode specifies the data transfer, arithmetic and logic, control, and floating point operations that need to be carried out by the processor. Length of an opcode varies both between and within different architecture types

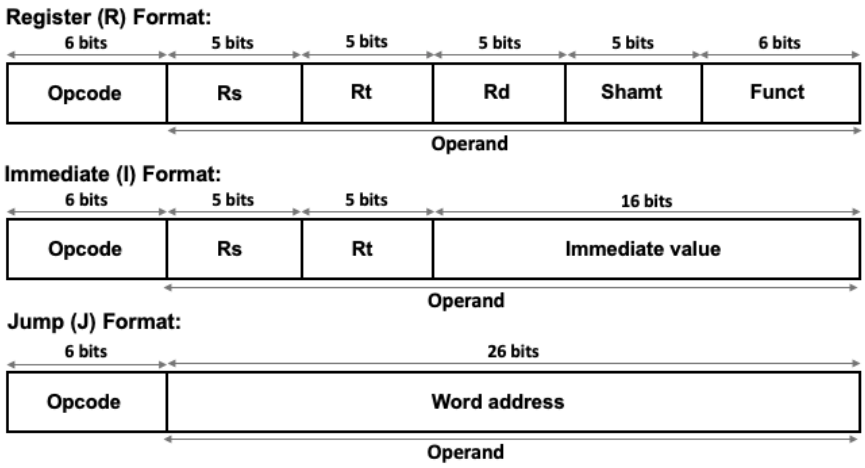


Figure 9.2: Three different 32-bit instruction formats of MIPS architecture [5]. Rs: Source register, Rt: Source/destination register, Rd: Destination register, Shamt: Shift values, Funct: Instructions to harware, Immediate value: Stores the constant values used in the immediate instructions (e.g., ADDI: add immediate). R format is used for the most arithmetic and logic instructions (e.g., ADD, XOR). I format is used for the data transfer, immediate and conditional branch instructions (e.g., MOVE, ADDI, BEQ: branch on equal). J format is used for unconditional jump instructions (e.g., JMP).

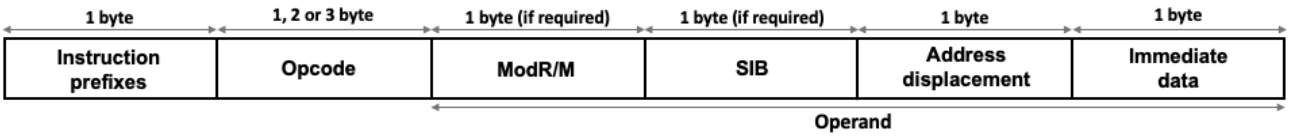


Figure 9.3: Variable length instruction format of X86_64 architecture [8]. ModR/M: used for memory addresses or opcode extensions, SIB: Scaled index byte. The length of an X86_64 instruction depends on the size of opcode or the usage of ModR/M and SIB.

(e.g., 4-bit ARM opcodes in Fig. 9.1 vs. 6-bit MIPS opcodes in Fig. 9.2 vs. 1,2 or 3-byte x86_64 opcodes in Fig. 9.3). The operand specifies the data values and memory/register addresses that are used in the operations specified by the opcode. The length of an operand depends on both architecture type and the type of data that it stores (e.g., 4-bit ARM destination register (Rd) address in Fig. 9.1 vs. 5-bit MIPS R format Rd address in Fig. 9.2 vs. 26-bit MIPS J format memory address in Fig. 9.2). Architectures have multiple instruction formats to support operations that use different

type and number of operands (e.g., R, I and J instruction formats in Fig. 9.2).

9.2.2 Endianness of an Instruction Set Architecture

The endianness defines the *byte order* of a multi-byte data variable representation. There are two types of endianness: (i) little-endian and (ii) big-endian. Under little-endian representation of a 2-byte number, the most-significant byte occupies the lowest memory address. In big-endian representation, the least-significant byte occupies the lowest memory address. For example, a data variable with value 1 is stored as 0x0100 and 0x0001 under little-endian and big-endian 2-byte representations, respectively. Examples of common little-endian ISAs include ARM, AVR, and x86_64. ISAs such as MIPS, PowerPC, and SPARC use big-endian representation.

9.2.3 N-gram Term Frequency-Inverse Document Frequency

N-gram, Term Frequency (TF), and Inverse Document Frequency (IDF) are widely used feature methods in Natural Language Processing (NLP) tasks such as text mining and auto-completion of sentences [11, 19, 108, 130]. We provide the definitions and discuss some of the characteristic properties of these NLP-based feature methods below.

Definition 9.2.1 (**N-gram**). An N-gram is a contiguous sequence of N terms (e.g., characters, words) in the content of a document (e.g., text message, news article). Typically, N-grams are extracted by moving a window of length N forward, one[2] term at a time, along the content of a document.

Definition 9.2.2 (**Term Frequency** [108]). The term frequency (TF) is the number of times each term appears in a document. Typically, TF values are divided by the total number of terms

[2]NLP applications that involves learning subject-verb relationships and word semantics in languages may require moving the window forward by multiple terms.

(normalized) in the document to mitigate the effect of the document length[3]. TF of a term τ is

$$\text{TF}(\tau) = \frac{\text{\# of times } \tau \text{ appeared in the document}}{\text{\# of terms in the document}}. \tag{9.1}$$

Definition 9.2.3 (**Inverse Document Frequency** [108]). The inverse document frequency (IDF) measures the informativeness of terms in a collection of documents (corpus). It assigns lower values to terms that commonly appear among the documents in the corpus as they do not contribute in distinguishing the contents of documents. Conversely, higher values are assigned to the less frequent terms in the corpus as they may constitute patterns inherent to the content of the documents. IDF of a term τ is

$$\text{IDF}(\tau) = \log\left(\frac{\text{\# of documents in the corpus} + 1}{\text{\# of documents with } \tau + 1}\right) + 1. \tag{9.2}$$

Definition 9.2.4 (**TF-IDF** [108]). The TF-IDF is a statistic that measures the importance of a term to a document in a corpus. TF-IDF value of a term τ is defined as:

$$\text{TF-IDF}(\tau) = \text{TF}(\tau) \times \text{IDF}(\tau) \tag{9.3}$$

In the context of NLP tasks, the N-gram TF-IDF presents a way to associate meaningful numerical values to words (i.e., scoring of words) that can be provided as the inputs to the ML models. A number assigned to a N-gram word in a document under N-gram TF-IDF method is increased proportionally by the number of times the corresponding N-gram word appears in the particular document, but is decreased by the number of documents that contain the word.

[3]Documents can have different lengths with regard to the number of terms recorded in them. In such scenario, a term τ may appear more frequently in longer documents compared to shorter documents. Hence unnormalized TF values of τ in longer documents will be higher than the values corresponding to shorter ones faulty indicating τ is more dominant pattern inherent to longer documents.

Therefore, words that are common in each document, such as “a", “the", “and", “this", “that", and “if" will have lower TF-IDF values even though they may appear frequently, as they do not help in identifying the content of a document in particular. However, if a specific N-gram word appears frequently in a document (or a smaller subset of documents), while not appearing frequently in others, it probably means that this particular N-gram word is very relevant to the content of the document (or the subset of documents). For example, if a 1-gram word “Computer" appears frequently in 20 web articles out of 1000, then it is likely that these 20 articles are related to computers.

9.2.4 Encoded Binaries

A binary-to-text encoding converts binary data to a sequence of printable characters. Such encodings are necessary for transmission of data when the communication channels do not allow binary data (such as email or Network News Transfer Protocol-NNTP). Rows III-VI in Table 9.2 below present examples of the binary code in Row I encoded into four popular binary-to-text encoding methods; Base16, Base32, Base64, and Base 85, respectively.

Under Base16, every byte of binary data is encoded to 2 characters. Base32 encodes 5-bytes of binary data into 8 characters. Base64 encodes 3-bytes of binary data into 4 characters. Lastly, Base85 encodes 4-bytes of data into 5 characters. The suffix of the encoding method name provides the number of distinct characters included in the alphabet of the encoding method. For example Base16 alphabet consists of 16 characters; digits $0-9$, and letters $A-F$. Additionally, Base32 and Base64 encoding use a padding character, “=".

9.3 Existing Binary Code Feature Extraction Techniques

In this section we first present an overview of binary code feature extraction techniques proposed in the literature for different applications. Then we present existing binary code features used for ML-based ISA identification and provide a brief review of cyber security applications that use the N-gram TF-IDF feature model.

Table 9.2 An example illustrating different binary files formats. Binary data in row I has been grouped into 8-bit values to represent 1-byte of data following the normal convention. Row II shows the hexadecimal (Hex) representation of the binaries in row I. Row III, Row IV, Row V, and Row VI present the Base16, Base32, Base64, and Base85 encoding of the binaries given in Row I, respectively.

	File format	Example
I	**Binary**	11010111 01000011 11010100 01000100 11010110 01000100 11011000 01000101
II	**Hex**	d7 43 d4 44 d6 44 d8 45
III	**Base16 encoded**	D743D444D644D845
IV	**Base32 encoded**	25B5IRGWITMEK===
V	**Base64 encoded**	10PURNZE2EU=
VI	**Base85 encoded**	f0e%UejS.Z

Feature extraction from binaries has been widely studied in the areas of ML-based malware detection [72, 117, 119] and file type identification [10, 70, 87]. The authors of [72] presented a feature extraction method for malware detection that uses information gain to select top 500, 4-gram byte patterns found in the training set binaries. The work in [117] proposed malware detection using a set of features composed with byte entropy histograms, string 2D histograms, and vectors corresponding to the hash values of binary file's metadata and import address tables. The authors of [119] combined the Boolean features related to the usage of dynamic-link library (DLL) files, function calls, GNU strings, and byte sequences to detect malware. The work in [87] introduced byte frequency distributions (BFDs) of object code and file header/trailer, and byte frequency cross-correlation distribution for file type identification. The authors of [70] suggested file type identification using a combination of the BFD and the frequency distribution corresponding to the rate of change of the byte content. The authors of [10] used cosine similarity of BFD features to predict the file type.

ML frameworks presented in [122] and [85] use information gain based top N 4-gram byte features for target architecture identification of firmware binaries. However, these approaches assume that a single instruction is stored in four bytes (32-bit) and hence are not suitable when identifying architectures that use different byte sizes (e.g., 8-bit, 16-bit, 64-bit) or variable length instructions (e.g., x86_64) to store instructions. Moreover, information gain computations require finding the number of different instruction set bit patterns recorded in each 4-gram byte pattern corresponding to each target architecture considered. Such a pattern searching procedure becomes computationally exhaustive when the number of target architectures is increased.

The author of [42] proposed to combine the byte-histogram features (i.e., BFD) with a set of features that capture endianness of binaries for ISA identification. Under this method, all the features are extracted from the decoded program binaries (files in binary or hex format). First for each binary file, a normalized byte-histogram is generated by counting all individual byte values in the file content. This provides 256 features. The endianness features are extracted by counting byte pairs which correspond to code sections – which increment by one (0x0001 vs 0x0100) – as well as those sections that correspond to a decrement by one (0xfffe vs 0xfeff). The authors of [21] adapted the byte-histogram along with endianness features introduced in [42] and proposed a simplified technique to determine the endianness of a binary file. Under the simplified endianness features, if it was found that there were more 0x0001's than 0x0100's then the entry of the feature vector corresponding to the big endianness was assigned a value of 1 and the entry corresponding to the little endianness was assigned 0. If it is found that the abundance is of the form 0x0100, then the reverse assignments are made.

Recently, the works [98] and [68] extended the byte-histogram and endianness features introduced in [42] by adding additional signature-based features extracted from the function epilogue and function prologue sections. Specifically, [98] introduced 31 new signature-based features extracted from binary files of amd64, arm, armel, mips32, powerpc, powerpc64, s390x and x86 ISAs. The authors of [68] introduced two additional signature-based features to identify powerpc ISAs. However, there is no guarantee that these signature-based features are included in partial binaries. As observed in [98], extending signature-based features to identify additional ISAs will require

significant effort and expert knowledge. Additionally, we note that the byte-histogram or BFD features are susceptible to frequent byte patterns commonly observed among binaries of different architectures (i.e., noisy byte patterns).

The N-gram TF-IDF feature model has been used in cyber security applications such as software vulnerability assessments [23, 142] and cyber threat detection [12, 37, 129, 143]. The authors of [23] used TF-IDF features extracted from bug reports to develop a tool for identifying software bugs. The work presented in [142] used TF-IDF features of Android application package's (Apk's) manifest file to evaluate the security of Android applications. The authors of [37] extracted TF-IDF features from process logs to build an intrusion detection system for a computer network. The research presented in [129] used TF-IDF features extracted from opcode sequences to classify ransomware families. The authors of [143] and [12] used TF-IDF features extracted from Application Programming Interface (API) call sequences for malware classification.

Table 9.3 Details of the (1,2,3)-gram TF-IDF features extracted from the different encoded binary file formats.

Binary file format	**Granularity of features**	**Number of features**			
		1-gram	**2-gram**	**3-gram**	**Total**
Base16 encoded	character	16	256	4096	**4368**
Base32 encoded	character	32	1024	5000	**6056**
Base64 encoded	character	64	4096	5000	**9160**
Base85 encoded	character	85	7225	5000	**12310**
Decoded (in Hex)	byte	256	65536	5000	**70792**

9.4 Natural Language Processing Techniques for Binary Code Feature Extraction

In this section we propose two object code feature selection methods named byte-level (1,2,3)-gram TF-IDF features (Section 9.4.1) and encoded character-level (1,2,3)-gram TF-IDF features (Section 9.4.2) for ML-based ISA identification. We first present the key observations that motivate

the N-gram TF-IDF structure of the two object code feature methods.

The accuracy of instruction set architecture identification largely depends on the ability of the set of object code features to capture bit patterns that help distinguish between different architectures. In Section 9.2.1, we observed that the opcodes and operands are two of the most important information embedded in binary instructions and they can have different lengths both within the instructions of same ISA and across the instructions of different ISAs. Hence, identifying a sufficient number of bit patterns that can enable high accuracy ISA identification across increased number of architectures is a non-trivial task that requires domain knowledge about the ISAs.

In what follows, we describe a method to characterize binary code features that does not require domain knowledge about the ISAs. Our approach involves the following steps: i) identify the smallest unit that has meaning (e.g., in NLP, this unit is a word) within the binaries; ii) identify the fixed-length patterns that need to be extracted from the binaries; iii) form a frequency vector of all possible lengths $1, 2, \ldots$ that can be used as the set of features for the binary files. We note that the frequency vector must satisfy the following properties:

1. Length of object code binaries should not influence[4] frequency values.

2. Values corresponding to frequent patterns that also commonly appear among the binary files (i.e., noisy patterns) should be attenuated as they do not help in distinguishing between the ISAs of the binaries.

3. Values corresponding to frequent patterns appearing only in a small subset of binary files should be boosted since such patterns will have higher probability of being ISA-specific patterns that can aid architecture identification.

We adapt N-gram TF-IDF feature model for extracting object code features. We propose two approaches to determine what will constitute a meaningful *term* in binaries and selecting an

[4]Consider any two binaries X and Y of same architecture type A with X having smaller length and Y having significantly larger object code length. In such scenario, X will be seen as much less type A compared to Y as the frequency of the architecture specific bit patterns in X will be always significantly less than the frequency of the same bit patterns in Y.

appropriate N for capturing architecture prominent patterns.

9.4.1 Byte-level N-gram TF-IDF Features

As noted in Section 9.2.1 an instruction recorded in a binary file is composed of a collection of consecutive bytes. Processors read and process each code section of a binary file byte-by-byte to execute instructions. Therefore, we first choose a byte as a term when adapting N-gram TF-IDF for extracting features from object code binaries.

Opcodes define architecture specific operations whereas operands define data and addresses that usually takes random values. Thus, the opcodes consist of more structured patterns that can characterize ISAs of binaries. For example in Section 9.2.1 we observed that ARM and MIPS architectures typically have opcodes of length of 4-bits and 6-bits (Fig. 9.1 and Fig. 9.2) respectively while opcodes of X86_64 can be either 1-byte, 2-byte or 3-bytes. This implies we can expect higher TF-IDF values for 1-gram byte patterns that include 4-bit and 6-bit opcode patterns in respective ARM and MIPS feature vectors. Similarly feature vectors of X86_64 binaries will have higher TF-IDF values for 1-gram, 2-gram or 3-gram byte patterns corresponding to 1-byte, 2-byte or 3-byte opcodes, respectively. Therefore, we extract 1-gram, 2-gram and 3-gram (i.e., $N = 1, 2,$ and 3) byte patterns from binaries and use a vector corresponding to their TF-IDF values as the binary object code features.

Further, 2-gram byte-level TF-IDF values allow capturing consecutive byte patterns in operands that are induced by the endianness property of an architecture (Section 9.2.2). For example, the data value 1 can be considered as a commonly used operand across the binaries as many object codes may include instructions related to increasing *for* and *while* loops variables by 1. Then TF-IDF feature vectors of big endian MIPS architecture binaries can expect to have higher values associated with the 0x0001 2-gram byte pattern compared to the values of 0x0100. On the other hand features of little endian ARM and X86_64 may have higher values for 0x0100 and relatively lower values for 0x0001.

Our proposed byte-level object code features include all $(1, 2)$-gram byte patterns and top 5000 ranked 3-gram byte-patterns. The rank ordering of the observed 3-gram byte-patterns are

done using the frequency of those patterns across all binary files in the training set binaries. We only include top 5000 ranked 3-gram byte-patterns since capturing all such patterns will require a large number of features ($256^3 >> 5000$) that can drastically increase the computation time and resources such as memory and processing power required for ML-based ISA identification. Therefore, byte-level (1,2,3)-gram TF-IDF features do not depend on a limited number of pre-selected byte patterns based on the domain knowledge and heuristics that may be completely absent in some (partial) binary data. Rather, our approach provides a more general set of expert agnostic features to identify byte patterns induced by opcodes and endianness of ISAs. There are $2^8 = 256$ possible 1-gram byte (8-bit) patterns. Hence, the total number of required feature values to represent each binary file under this method can be as large as $256 + 256^2 + 5000 = 70792$.

9.4.2 Character-level N-gram TF-IDF Features from Encoded Binaries

As discussed in Section 9.4.1, bit-patterns of opcodes play a more vital role in identifying ISAs since they contain operations inherent to the target processor architecture. In addition, we observe that opcode information embedded in the instructions are typically less than 1-byte (e.g., 4-bit opcodes in ARM, 6-bit opcodes in MIPS, AVR, and PowerPC) with the exception of 1,2, or 3-byte opcodes in X86_64. We also observe that there are other bit patterns of different lengths recorded in instructions (outside the bit patterns related to opcodes and operands) that may be used to distinguish between architecture types. Examples include 4-bit condition field in ARM instructions which is (Fig. 9.1) mostly set to 1110 for indicating “always execute" and 1-byte instruction prefixes in X86_64 instructions (e.g., 0xf0 - repeat/lock prefix; 0xf2 and 0xf3 - string manipulation prefixes). Hence, accurately capturing such fine-grained bit patterns specific to architectures requires choosing terms with less than 8-bits (ideally 4, 5 or 6-bit terms).

As noted in Section 9.2.4, encoding methods naturally provide a way to group binary data into different length bits. Therefore, we propose to extract character-level (1,2,3)-gram TF-IDF features from encoded binaries to enable capturing architecture specific fine grained bit patterns. Table 9.3 provides the number of features we used under each encoding method and their composition. Using this method allows us to reduce the required number of features by approximately $16\times$ compared

Table 9.4 Details of histogram + endianness features extracted from different binary file formats.

Binary file format	Granularity of features		Number of features		
	Histogram	**Endianness**	**Histogram**	**Endianness**	**Total**
Base16 encoded	character	2-byte	16	4	**20**
Base32 encoded	character	2-byte	32	4	**36**
Base64 encoded	character	2-byte	64	4	**68**
Base85 encoded	character	2-byte	85	4	**89**
Decoded (in Hex)	byte	2-byte	256	4	**260**

to byte-level features discussed in Section 9.4.1.

9.5 Experiment Setup

This section presents the details of the experiments used to compare the performance of ML algorithms for ISA identification under two types of feature selection methods: 1) Histogram along with Endianness (Hist. + Endian) features and 2) (1,2,3)-gram TF-IDF features. First, we detail the characteristics of the datasets used in our experiments. Then we present the properties of the different types of features extracted from the datastes. All the experiments are implemented using Python 3.8.5 on a workstation with Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz processor and 128 GB memory.

9.5.1 Datasets of Object Code Binaries

Praetorian Dataset: This dataset consists of $202,066$ distinct Base64 encoded binaries downloaded from Praetorian's "Machine Learning Binaries" challenge web page [9]. We use this dataset as our primary dataset since [9] provides a user friendly Application Programming Interface (API) that allows on-demand download of unlimited number of equal length binaries. Each encoded

binary string in this dataset consists of exactly 88 characters (66 bytes) and belongs to one of the following 12 architecture types: avr, alphaev56, arm, m68k, mips, mipsel, powerpc, s390, sh4, sparc, x86_64, and xtensa. We divided the primary dataset into 50 non-overlapping subdatasets. In order to mitigate any training biases that can arise from data imbalance, we ensured each subdataset contained equal number of binaries per ISA. Each subdataset was further partitioned into a training set with 2856 encoded binaries (238 per ISA) and a testing set with 960 encoded binaries (80 per ISA).

We applied a series of decoding and encoding operations to the Base64 encoded binaries of each subdataset to create additional subdatasets of four other data formats: binary, Base16, Base32, and Base85. The 8-bit (byte) values in binary formatted subdatasets were further converted to their corresponding 2-character Hex values for ease of analysis.

Object Code Dataset: This dataset was downloaded from [7]. It consists of $66,685$ binaries extracted from the object code section of ELF binary files. Binaries in this dataset belong to one of the following 23 architecture types: alpha, amd64, arm64, armel, armhf, hppa, i386, ia64, m68k, mips, mips64el, mipsel, powerpc, powerpcspe, ppc64, ppc64el, riscv64, s390, s390x, sh4, sparc, sparc64, and x32. We divided the dataset into 4 non-overlapping subdatasets. However, object code dataset is imbalanced and the size of each binary is varied from 2 kB to 64 kB. Therefore, we mitigate the data imbalance problem via choosing equal number of binaries per ISA for each subdataset. We extracted only the first 128 bytes from each binary to make them equally sized. Each subdataset was further partitioned into a training set with 7130 encoded binaries (310 per ISA) and a testing set with 1771 encoded binaries (77 per ISA).

We applied a series of encoding operations to the binaries of each subdataset to create additional subdatasets of four other data formats: Base16, Base32, Base64, and Base85. As in Praetorian Dataset, the 8-bit (byte) binaries in the subdatasets were further converted to their corresponding 2-character Hex values for ease of analysis.

9.5.2 Baseline Object Code Features

We use the byte-histogram and endianness features introduced in the seminal work [42] for ISA identification as our baseline for byte-level (1,2,3)-gram TF-IDF features. Majority of the related work including [21,85,98,122], and [68] use byte-histogram and endianness features as the baseline method to compare. Moreover, byte-histogram and endianness features only use domain knowledge to synthesize four features to capture the endianness of ISAs. Therefore, it is well suited as the baseline for our proposed feature models that does not require any expert knowledge on ISAs for extracting features from the binaries.

The byte-histogram features of each binary are extracted by counting the number of times each distinct Hex value appears in the decoded binary. The byte-histogram of each binary is then normalized by the total number of Hex values recorded in the corresponding binary. Note that the byte-histogram features are equivalent to the 1-gram TF features and they form the first $2^8 = 256$ entries in the feature vector. Then four more domain knowledge/heuristic-based features are added to the feature vector for capturing the endianness. These additional features are extracted by counting the number of times each 2-byte Hex values, 0x0001, 0x0100, 0xfffe, and 0xfeff appear in each binary. These counts are also normalized by the total number of Hex values in the binary.

In order to evaluate the effectiveness of our character-level (1,2,3)-gram TF-IDF features extracted from the encoded binaries, we use character-histogram and endianness features. The character-histogram features of each binary are extracted by counting the number of times each distinct character appears in the encoded binary. The character-histogram of each encoded binary is then normalized by the total number of characters recorded in the corresponding binary. Since the domain knowledge/ heuristic-based four endianness features introduced in [42] are based on the 2-byte Hex values, we extract the four endianness features from the decoded binaries in Hex format following the same steps as in byte-histogram and endianness features. Table 9.4 summarizes the details about the *histogram + endianness* features.

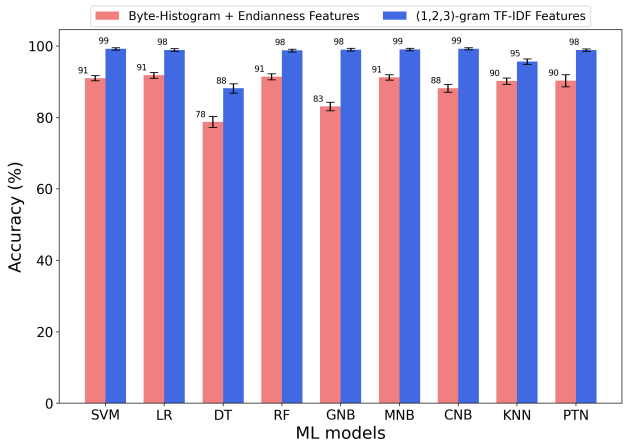


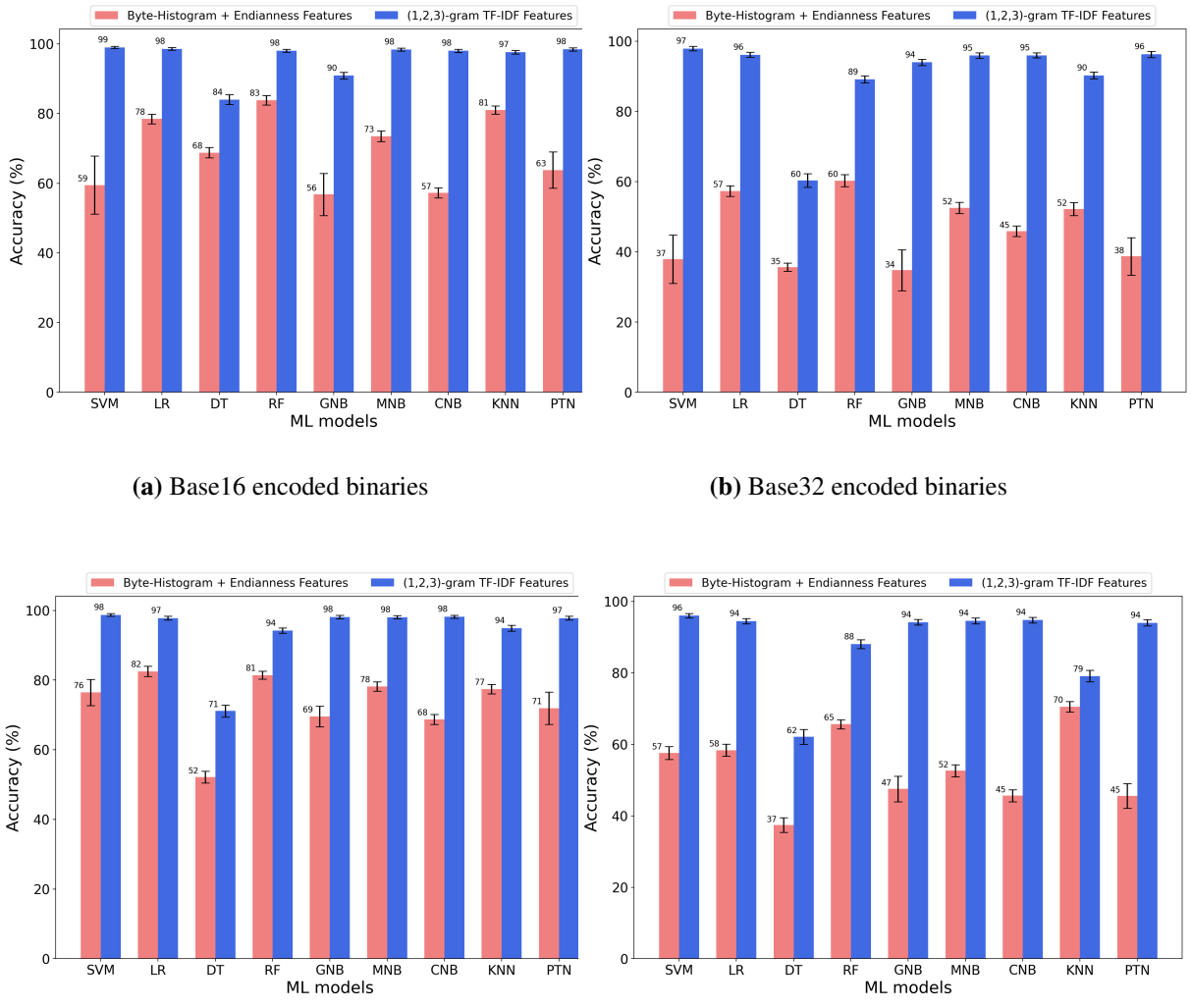
Figure 9.4: Accuracy of instruction set architecture identification under different machine learning algorithms corresponding to byte-histogram along with endianness features and byte-level (1,2,3)-gram TF-IDF features extracted from decoded binaries of Praetorian dataset. Each accuracy and error bar value is computed across 50 uniformly distributed independent subdatasets. The byte-level (1,2,3)-gram TF-IDF features consistently results in a higher accuracy compared to the byte-histogram + endianness features.

9.6 Results and Discussions

In this section, we present the experimental results and related discussions. We use the following abbreviations to denote the different ML algorithms used in the experiments. SVM: Support Vector Machine, LR: Logistic Regression, DT: Decision Tree, RF: Random Forest, GNB: Gaussian Naive Bayes, MNB: Multinomial Naive Bayes, CNB: Complement Naive Bayes, KNN: K-Nearest Neighbor, and PTN: Perceptron. We use *histogram + endianess* to refer to the baseline features. We refer to [57] for the detailed descriptions about the aforementioned ML algorithms.

9.6.1 Evaluation Results on Praetorian Dataset

Accuracy of the ML models: Fig. 9.4 compares the accuracy of instruction set architecture identification under different ML algorithms corresponding to byte-histogram + endianness features and byte-level (1,2,3)-gram TF-IDF features extracted from the decoded binaries of Praetorian dataset. Accuracy values that we report in our experiments are averaged over 50 uniformly distributed



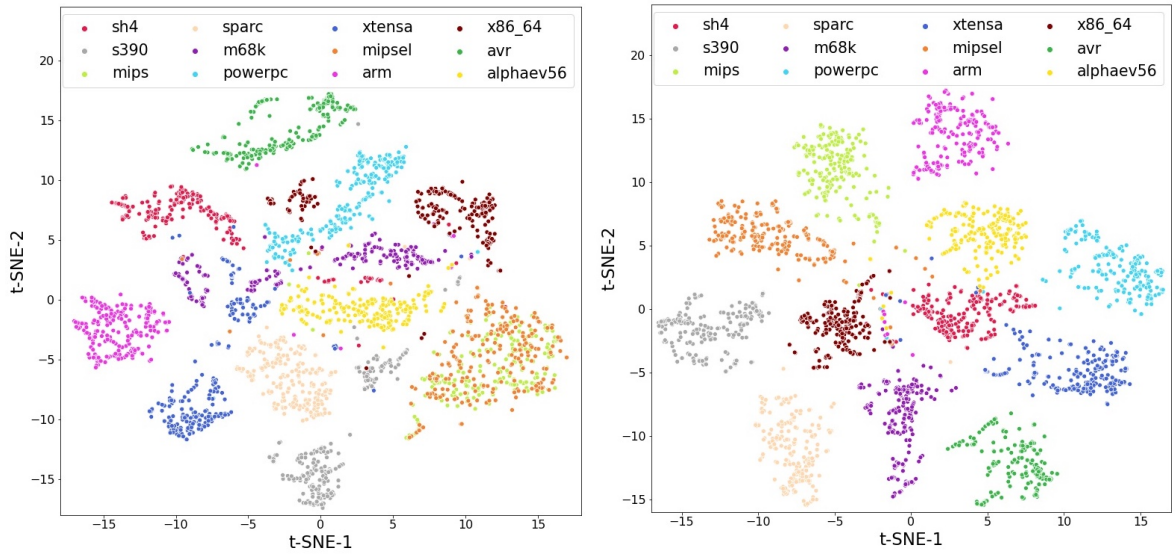
(a) Base16 encoded binaries

(b) Base32 encoded binaries

(c) Base64 encoded binaries

(d) Base85 encoded binaries

Figure 9.5: Accuracy of 12 architecture program binary classification of different machine learning algorithms corresponding to character-histogram along with endianness features and character-level (1,2,3)-gram TF-IDF features under different encoded binary file formats of Praetorian dataset, Base16, Base32, Base64, and Base85. Each accuracy and error bar value is computed across 50 uniformly distributed independent subdatasets. The character-level (1,2,3)-gram TF-IDF features consistently results in a higher accuracy compared to the histogram + endianness features.



(a) Byte-histogram + Endianness features from decoded binaries

(b) Byte-level (1,2,3)-gram TF-IDF features from decoded binaries

Figure 9.6: 2-D t-SNE plots corresponding to byte-histogram + endianness features (Fig. 9.6-(a)) and byte-level N-gram TF-IDF features (Fig. 9.6-(b)) extracted from randomly chosen training datasets consist of decoded binaries. The byte-level (1,2,3)-gram TF-IDF features provide better separability compared to the byte-histogram + endianness features.

independent subdatasets that contain binaries from 12 different architectures. Our results show that using byte-level (1,2,3)-gram TF-IDF features increase the accuracy values by $8.78\% \pm 2.71\%$ on average across the ML models considered. Maximum and minimum accuracy increments are 15% at GNB and 5% at KNN, respectively. The proposed feature method yields the highest accuracy of 99% at SVM, MNB, and CNB while the baseline achieve the highest accuracy of 91% at SVM, LR, RF, and MNB.

Properties of byte-level (1,2,3)-gram TF-IDF features such as ability to suppress the effects of noisy bytes, providing more generalized set of features to capture the architecture related characteristics such as endianness, and providing increased number of features that can capture byte patterns specific to architectures enable consistently achieving higher levels of accuracy. However, the high

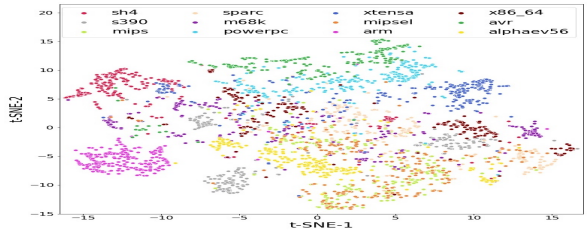
accuracy is achieved at the expense of considering around $200\times$ more features than the baseline (compare last rows of Table 9.3 and Table 9.4).

Fig. 9.5 illustrates the accuracy of different ML algorithms corresponding to character-histogram + endianness features and character-level (1,2,3)-gram TF-IDF features when base16, base32, base64, and base85 encoded binaries of Praetorian dataset are used. Our results show that character-level (1,2,3)-gram TF-IDF features has a higher accuracy than the character-histogram + endianness features across all ML models. The best accuracy of 99% is achieved when character-level TF-IDF features are extracted from the Base16 binaries and the SVM model is used for identifying ISAs.

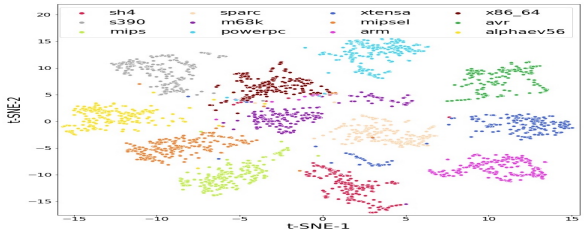
Character-level encoded (1,2,3)-gram TF-IDF features provide the noise reduction and increased generalizability advantages of (1,2,3)-gram TF-IDF features. It also provides increased number of fine grained features to capture bit patterns specific to architectures that are smaller than 8-bits (e.g., nibble patterns of opcodes). More importantly, using encoded character-level (1,2,3)-gram TF-IDF features requires only $\sum_{i=1}^{3} 16^i = \mathbf{4368}$ features compared to $\sum_{i=1}^{2} 256^i + 5000 = 70792$ features corresponding to the byte-level (1,2,3)-gram TF-IDF features.

Quality of the features via t-SNE: Fig. 6 compares the 2-D t-SNE representations of histogram + endianness features and N-gram TF-IDF features of the decoded binaries. Our experiments suggest that byte-level N-gram TF-IDF features result in better separation of the data points corresponding to the different architectures. In contrast, using histogram + endianness features lead to data points of mips and mipsel architectures being indistinguishable. This explains the high classification accuracy achieved using the proposed byte-level N-gram TF-IDF features.

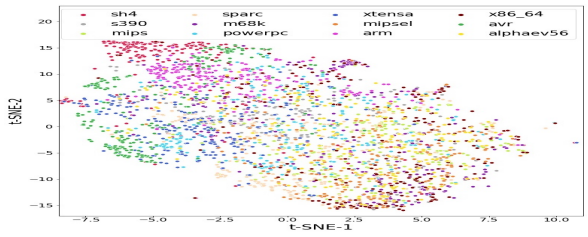
Fig. 7 shows the 2-D t-SNE representations of histogram + endianness features and N-gram TF-IDF features of the base16, base32, base64, and base85 encoded binaries. Our experiments show that baseline histogram + endianness features results in poor separation of the clusters related to the data points corresponding to the different architectures. On the other hand, N-gram TF-IDF features extracted from Base16 encoded binaries provide a better separation for the clusters related to different architectures. In fact, comparing with Fig. 6, we can observe this separation is slightly better than the separation achieved via byte-level N-gram TF-IDF features.



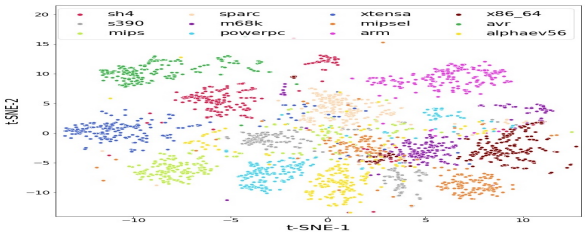
(a) Char-hist + Endianness features of Base16 binaries



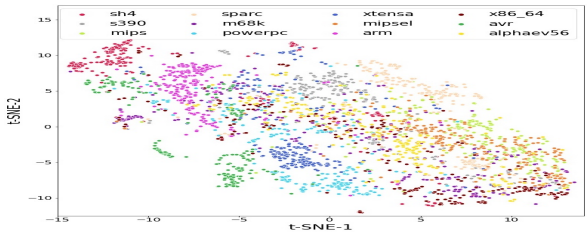
(b) Char-level (1,2,3)-gram TF-IDF features of Base16



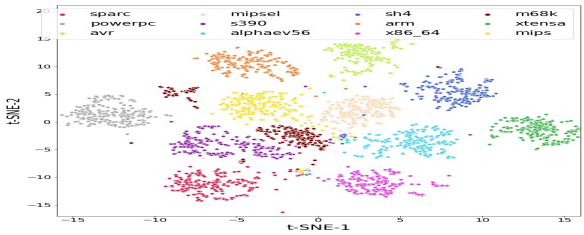
(c) Char-hist + Endianness features of Base32 binaries



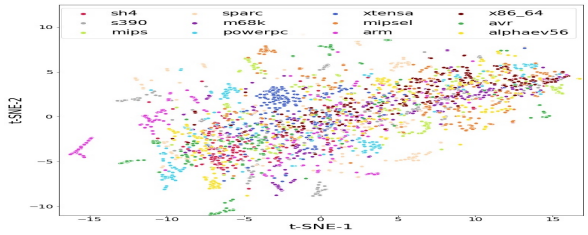
(d) Char-level (1,2,3)-gram TF-IDF features of Base32



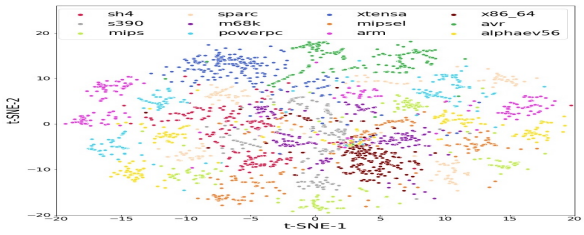
(e) Char-hist + Endianness features of Base64 binaries



(f) Char-level (1,2,3)-gram TF-IDF features of Base64



(g) Char-hist + Endianness features of Base85 binaries



(h) Char-level (1,2,3)-gram TF-IDF features of Base85

Figure 9.7: 2-D t-SNE plots corresponding to byte-histogram + endianness features and character-level N-gram TF-IDF features extracted from randomly chosen training datasets consist of Base16, Base32, Base64, and Base85 encoded binaries. The character-level (1,2,3)-gram TF-IDF features provide better separability compared to the character-histogram + endianness features.

Number of training data required: Fig. 9.8 and Fig. 9.9 plot accuracy values against the number of training data when classifying different number of ISAs using the Support Vector Machine (SVM) and Logistic Regression (LR) ML models, respectively. These experiments use byte-level (1,2,3)-gram TF-IDF features extracted from decoded binaries. Our results show that byte-level (1,2,3)-gram TF-IDF features achieve high accuracy in both SVM ($>$ 98 %) and LR ($>$ 97%) consistently under all architecture scenarios. Moreover, in SVM only around 1000 binaries (84 binaries per architecture) are required in the training data set to achieve accuracy $>$ 98 %. In the case of LR, 1300 binaries (103 binaries per architecture) are required in the training data set to achieve accuracy $>$ 97 %. This shows that byte-level (1,2,3)-gram TF-IDF features does not require large number of training data to achieve high accuracy. Similar results can be observed in the case of character-level (1,2,3)-gram TF-IDF features extracted from encoded binaries.

9.6.2 Evaluation Results on Object Code Dataset

Fig. 9.10 compares the accuracy of ISA identification under different ML algorithms corresponding to byte-histogram + endianness features and byte-level (1,2,3)-gram TF-IDF features extracted from the binaries in the object code dataset. Accuracy and error bar values given in the figure are averaged over 4 uniformly distributed independent subdatasets that contain binaries from 23 different architectures. The results convey that using byte-level (1,2,3)-gram TF-IDF features increase the accuracy values by $6.67\% \pm 3.27\%$ on average across the ML models considered. The maximum increment of 13% is attained at GNB and the minimum increment of 4% is obtained at KNN. Moreover, our proposed method achieves a top accuracy of 98% at SVM, LR, RF, and PTN. The baseline achieves its top accuracy of 94% at RF.

Fig. 9.11 shows the accuracy of different ML algorithms corresponding to character-histogram + endianness features and character-level (1,2,3)-gram TF-IDF features when base16, base32, base64, and base85 encoded binaries in the object code dataset are used. Our results indicate that using character-level (1,2,3)-gram TF-IDF features yield higher accuracy across all the ML models in comparison with using the character-histogram + endianness features. The best accuracy of 98%

is achieved when the proposed feature model is used on all four types of encoded binaries with the SVM model. Similar results is also observed when the proposed character-level TF-IDF features are used on Base64 encoded binaries with PTN model.

Additionally, we observe that on average both proposed and baseline feature extraction methods perform better on object code dataset compared to their respective performance on Praetorian dataset even with object code dataset having 23 different ISAs in the dataset compared to the 12 distinct ISAs in Praetorian dataset. Specifically, the performance of baseline feature extraction methods from encoded binaries improves by a large margin from Praetorian dataset to object code dataset. We presume such phenomenon occurs due to the fact that a much larger size of binaries are used in the feature extraction process of object code binaries compared to the Praetorian binaries (128 bytes vs. 66 bytes).

We note that the results corresponding to the proposed methods in Fig. 9.10 and Fig. 9.11 are consistent with the results in Fig. 9.4 and Fig. 9.5 related to Praetorian Dataset. On the other hand, we observe that the baseline methods can have inconsistent performance heavily depended on the size of the binaries when they are applied on the encoded binaries. Overall results show that our proposed feature models can successfully detect the ISA of object code binaries extracted from real-world binary files in both binary and encoded formats while outperforming the baseline binary code feature extraction methods. Also, our TF-IDF feature extraction models can perform well even with the smaller sized binaries.

Remark 9.6.1. We note that ML-based ISA classification is susceptible to adversarial attacks. Specifically, it has been shown that the histogram (1-gram) and 2-gram based Intrusion Detection Systems (IDSs) are highly vulnerable to adversarial attacks such as mimicry attacks [135] and polymorphic blending attacks [51], respectively. Related work in the literature that includes but not limited to [55, 136] suggest that combinations of higher order N-grams (e.g., (3, 4, 5)-gram TF-IDF) are much more resilient against adversarial attacks compared to their standalone 1-gram or 2-gram counterparts. Recent work in [81] also shows mixture of N-gram TF-IDF features can be used to build Android malware detection system that is resilient to adversarial examples. Also, work in [128] shows that combining N-gram TF-IDF features with other NLP techniques such as

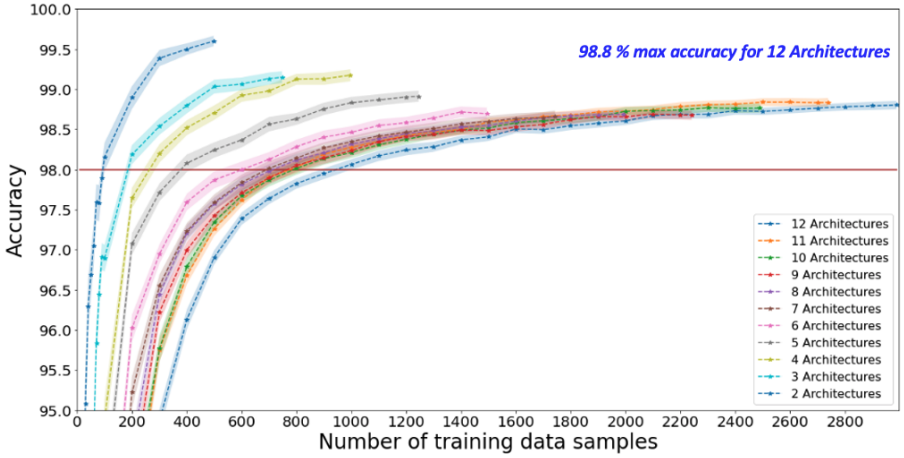


Figure 9.8: SVM accuracy results under different number of architectures when (1,2,3)-gram TF-IDF features are used. The red line indicates the 98% accuracy margin.

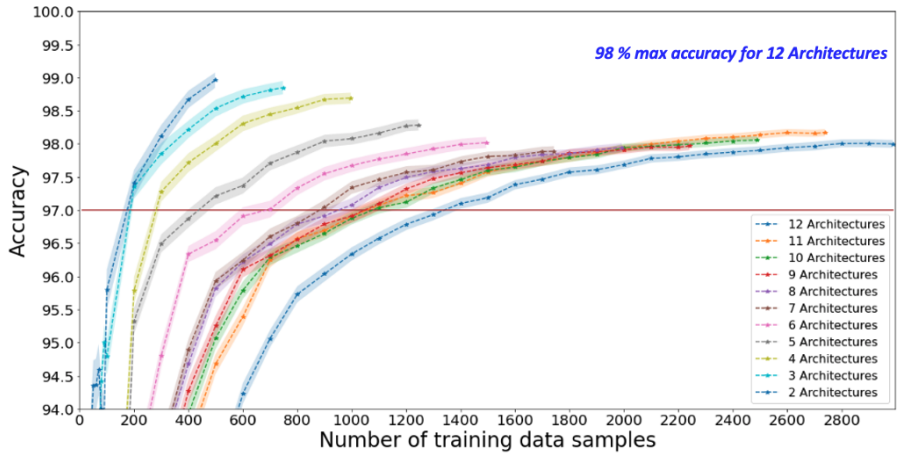


Figure 9.9: LR accuracy results under different number of architectures when (1,2,3)-gram TF-IDF features are used. The red line indicates the 97% accuracy margin.

transformers provide a set of resilient features against adversarial attacks on NLP tasks. We leave the study of the resiliency of our proposed feature extraction methods against adversarial examples as a promising future work.

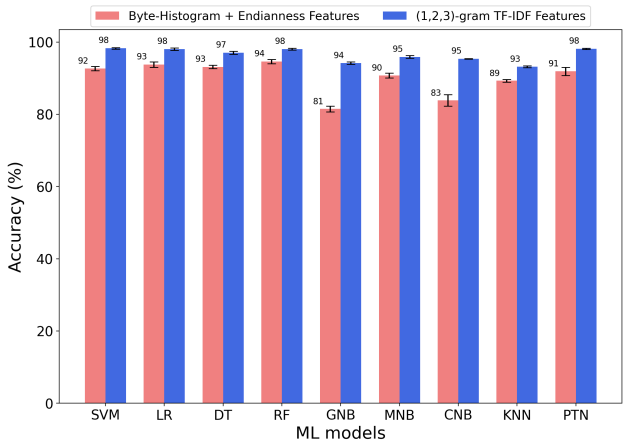
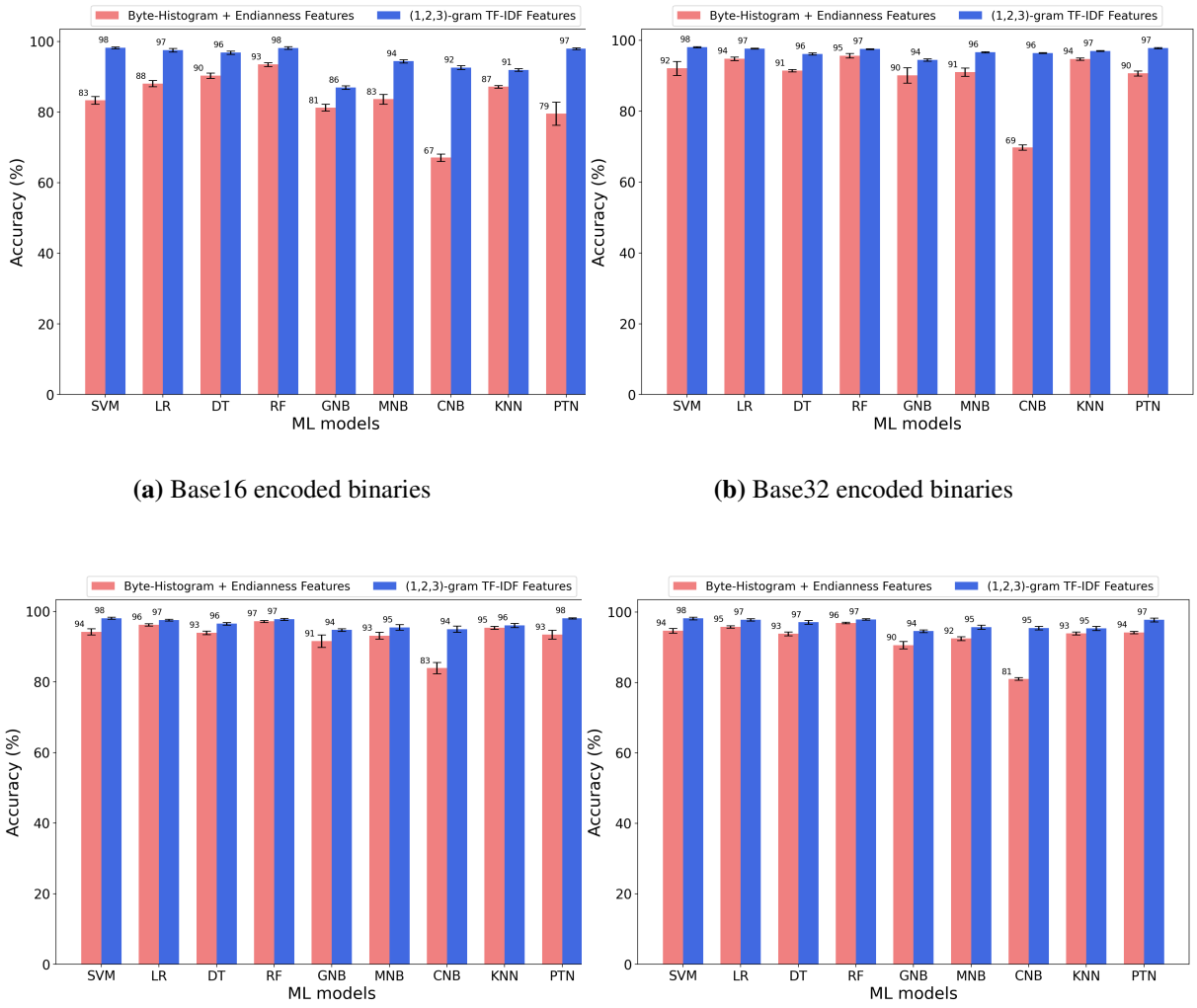


Figure 9.10: Accuracy of instruction set architecture identification under different machine learning algorithms corresponding to byte-histogram along with endianness features and byte-level (1,2,3)-gram TF-IDF features extracted from decoded binaries of object code dataset. Each accuracy and error bar value is computed across 4 uniformly distributed independent subdatasets. The byte-level (1,2,3)-gram TF-IDF features consistently results in a higher accuracy compared to the byte-histogram + endianness features.

9.7 Summary

In this chapter we proposed binary object code feature extraction methods based on N-gram Term Frequency-Inverse Document Frequency (TF-IDF) feature model for instruction set architecture (ISA) identification. We used byte-level N-gram TF features to extract the successive bytes patterns inherent to architectures. Setting $N = 1$, can recover a class of object code features used in the literature called byte histogram features. However, such approaches require additional domain knowledge/heuristic-based features for capturing successive byte patterns inherent to ISAs and they may be absent in partial binaries. Histogram-based features are also susceptible to noisy byte values. Hence, histogram and signature-based approaches fail to achieve high accuracy for the binaries with limited data that is corrupted by noise. We scaled byte-level N-gram TF features by their respective IDF values to attenuate the effect of noisy byte data. Using two different datasets containing binaries from 12 ISAs and 23 ISAs, we showed byte-level (1,2,3)-gram TF-IDF features are adequate to achieve high accuracy performance in the Machine Learning (ML)-based ISA



(a) Base16 encoded binaries

(b) Base32 encoded binaries

(c) Base64 encoded binaries

(d) Base85 encoded binaries

Figure 9.11: Accuracy of 23 architecture program binary classification of different machine learning algorithms corresponding to character-histogram along with endianness features and character-level (1,2,3)-gram TF-IDF features under different encoded binary file formats of object code dataset, Base16, Base32, Base64, and Base85. Each accuracy and error bar value is computed across 4 uniformly distributed independent subdatasets. The character-level (1,2,3)-gram TF-IDF features consistently results in a higher accuracy compared to the histogram + endianness features.

identification models.

We observed instruction bytes have architecture specific fine grained bit patterns and extracted such patterns using character-level N-gram TF-IDF features of encoded binaries (e.g., base16, base32, base64, base85). We observed character-level (1,2,3)-gram TF-IDF features of encoded binaries achieving high accuracy while only using less number of features (up to $\approx 16\times$) compared to the byte-level (1,2,3)-gram TF-IDF features. Our binary code feature extraction methods do not require any prior domain specific knowledge on the ISAs and hence, easily extendable to ISA identification with different number of ISAs. Promising future research directions include investigation of the effect of NLP and binary-to-text encoding-based object code feature extraction methods in the fields of file type identification and malware binary detection.

Chapter 10

CONCLUSIONS AND FUTURE DIRECTIONS

In this dissertation proposal we studied the problem of resource efficient and effective detection of Advanced Persistent Threats (APTs) using Dynamic Information Flow Tracking (DIFT) detection mechanism. Making use of the system log data information and postmortem/offline analysis of the data, we constructed the Information Flow Graph. We then modeled the strategic interactions between DIFT and APT as a nonzero-sum game on IFG. In our initial studies, we assumed DIFT does not incur any false-negatives and false-positives when performing a security analysis on a tagged information flow.

In Chapter 3, we modeled DIFT-APT game for detecting single-stage attacks and developed algorithms to identify the best set of system locations for tagging information flows that incur minimum resource overhead and enable high probability of APT detection for a set of predefined traps. We extended the DIFT-APT game model to incorporate multi-stage APT attacks in Chapter 4. We computed a Nash equilibrium for single stage attacks using a min-cut problem. We provided a polynomial-time algorithm to compute a correlated equilibrium for the multi-stage attack case. In Chapter 5, we modeled the simultaneous detection of multiple APTs using resource constrained DIFT. We proved the best response of defender and adversary are equivalent to maximizing an increasing DR-submodular function and solving a shortest path problem, respectively.

In Chapter 6, we incorporated the false-negatives and false-positives associated with the DIFT using a stochastic game framework. In Chapter 7, we presented a model-free, actor-critic algorithm to compute an NE of the discounted stochastic DIFT-APT game when the underlying false negatives and false positives of the DIFT are unknown. In Chapter 8, we formulated the interactions between DIFT and APT as an average reward stochastic game to capture the long-term behavior of the APTs and proposed a reinforcement learning algorithm, RL-ARNE, to learn an ARNE of an average

reward stochastic game. In Chapter 9, we proposed N-gram TF-IDF-based feature extraction methods for of Instruction Set Architecture (ISA) identification using partial binaries to facilitate DIFT in detecting known malicious patterns recorded in the program binaries.

We list the following research directions as promising future work:

- **Interpreting a Nash equilibrium of average-reward stochastic DIFT-APT game**: In Chapter 4, we interpret the Nash equilibrium of DIFT-APT game using graph theoretic results, i.e., min-cut problem. Finding a similar interpretable (sufficient) conditions for characterizing a Nash equilibrium of average-reward stochastic DIFT-APT game may help in developing fast converging equilibrium seeking algorithms.

- **Integrating detection into DIFT:** The simulation studies included in this dissertation used a set of handpicked false-negatives and false-positives of DIFT to evaluate the proposed approaches. Cyber threat intelligence report databases such as MITRE ATT&CK [6] and CVE (Common Vulnerabilities and Exposures) [2] publish signatures of the known APTs that can be mapped to the events described by the node-edge relations recorded in the IFG of the system. On the other hand, graph learning research provides a rich set of relational data embedding mechanisms such as Node2Vec [54], Graph2Vec [93], and translation-based embedding (e.g., TransE [28]) that can be leveraged to detect APTs. We will investigate how to incorporate these signature-based and machine learning-based APT detection mechanisms to model and realistically simulate DIFT's detection.

BIBLIOGRAPHY

[1] Arm instruction set.

[2] CVE. `https://cve.mitre.org`.

[3] Ghidra.

[4] Ida pro.

[5] Mips32 architecture.

[6] MITRE ATT&CK. `https://attack.mitre.org/`.

[7] Research dataset finder.

[8] x86-64 instructions set.

[9] Tech challenge: Machine learning binaries, Feb 2021.

[10] Irfan Ahmed, Kyung-suk Lhee, Hyunjung Shin, and ManPyo Hong. Content-based file-type identification using cosine similarity and a divide-and-conquer approach. *IETE Technical Review*, 27(6):465–477, 2010.

[11] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[12] Muhammad Ali, Stavros Shiaeles, Gueltoum Bendiab, and Bogdan Ghita. Malgra: Machine learning and n-gram malware feature extraction and detection system. *Electronics*, 9(11):1777, 2020.

[13] Tansu Alpcan and Tamer Başar. *Network Security: A Decision and Game-Theoretic Approach*. Cambridge University Press, 2010.

[14] Rabah Amir. Stochastic games in economics and related fields: an overview. In *Stochastic Games and Applications*, pages 455–470. Springer, 2003.

[15] Dennis Andriesse. *Practical Binary Analysis: Build Your Own Linux Tools for Binary Instrumentation, Analysis, and Disassembly*. no starch press, 2018.

[16] Gürdal Arslan and Serdar Yüksel. Decentralized Q-learning for stochastic teams and games. *IEEE Transactions on Automatic Control*, 62(4):1545–1558, 2016.

[17] Robert J Aumann. Correlated equilibrium as an expression of Bayesian rationality. *Econometrica: Journal of the Econometric Society*, pages 1–18, 1987.

[18] Francis Bach. Submodular functions: From discrete to continuous domains. *Mathematical Programming*, pages 1–41, 2016.

[19] Prafulla Bafna, Dhanya Pramod, and Anagha Vaidya. Document clustering: Tf-idf approach. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 61–66. IEEE, 2016.

[20] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[21] Bryan Beckman and Jed Haile. Binary analysis with architecture and code section detection using supervised machine learning. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 152–156. IEEE, 2020.

[22] Chandan Kumar Behera and D Lalitha Bhaskari. Different obfuscation techniques for code protection. *Procedia Computer Science*, 70:757–763, 2015.

[23] Diksha Behl, Sahil Handa, and Anuja Arora. A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf. In *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, pages 294–299. IEEE, 2014.

[24] Boldizsár Bencsáth, Gábor Pék, Levente Buttyán, and Mark Felegyhazi. The cousins of Stuxnet: Duqu, Flame, and Gauss. *Future Internet*, 4(4):971–1003, 2012.

[25] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to Linear Optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[26] Andrew An Bian, Baharan Mirzasoleiman, Joachim M Buhmann, and Andreas Krause. Guaranteed non-convex optimization: Submodular maximization over continuous domains. *International Conference on Artificial Intelligence and Statistics*, pages 111–120, 2017.

[27] Daniel Bilar. Opcodes as predictor for malware. *International journal of electronic security and digital forensics*, 1(2):156–168, 2007.

[28] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[29] Vivek S Borkar. Stochastic approximation with two time scales. *Systems & Control Letters*, 29(5):291–294, 1997.

[30] Christopher Brant, Prakash Shrestha, Benjamin Mixon-Baca, Kejun Chen, Said Varlioglu, Nelly Elsayed, Yier Jin, Jedidiah Crandall, and Daniela Oliveira. Challenges and opportunities for practical and effective dynamic information flow tracking. *ACM Computing Surveys (CSUR)*, 55(1):1–33, 2021.

[31] Guillaume Brogi and Valérie Viet Triem Tong. TerminAPTor: Highlighting advanced persistent threats through information flow tracking. *IFIP International Conference on New Technologies, Mobility and Security*, 2016.

[32] Niv Buchbinder, Moran Feldman, Joseph Seffi, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing*, 44(5):1384–1402, 2015.

[33] Gerardo Canfora, Francesco Mercaldo, Corrado Aaron Visaggio, and Paolo Di Notte. Metamorphic malware detection using code metrics. *Information Security Journal: A Global Perspective*, 23(3):57–67, 2014.

[34] Capstone. The ultimate disassembly framework, May 2020.

[35] Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, Learning, and Games*. Cambridge university press, 2006.

[36] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In *IFIP International Conference on Communications and Multimedia Security*, pages 63–72. Springer, 2014.

[37] Rung-Ching Chen and Su-Ping Chen. Intrusion detection using a hybrid support vector machine based on entropy and tf-idf. *International Journal of Innovative Computing, Information, and Control (IJICIC)*, 4(2):413–424, 2008.

[38] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM*, 56, 2009.

[39] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player nash equilibria. *Journal of the ACM*, 56(3):14, 2009.

[40] KR1442 Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[41] James Clause, Wanchun Li, and Alessandro Orso. Dytan: a generic dynamic taint analysis framework. *International Symposium on Software Testing and Analysis*, pages 196–206, 2007.

[42] John Clemens. Automatic classification of object code using machine learning. *Digital Investigation*, 14:S156–S162, 2015.

[43] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press: Cambridge, 2001.

[44] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *2018 IEEE symposium on security and privacy (SP)*, pages 161–175. IEEE, 2018.

[45] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. The complexity of computing a nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

[46] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of software: Evolution and Process*, 25(1):53–95, 2013.

[47] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems*, 32(2):5, 2014.

[48] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[49] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[50] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer Science & Business Media, 2012.

[51] Prahlad Fogla, Monirul I Sharif, Roberto Perdisci, Oleg M Kolesnikov, and Wenke Lee. Polymorphic blending attacks. In *USENIX security symposium*, pages 241–256, 2006.

[52] Dean Foster and Peyton Young. Stochastic evolutionary game dynamics. *Theoretical Population Biology*, 38(2):219, 1990.

[53] Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[54] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[55] Dina Hadžiosmanović, Lorenzo Simionato, Damiano Bolzoni, Emmanuele Zambon, and Sandro Etalle. N-gram against the machine: On the feasibility of the n-gram network analysis for binary protocols. In *Research in Attacks, Intrusions, and Defenses: 15th International Symposium, RAID 2012, Amsterdam, The Netherlands, September 12-14, 2012. Proceedings 15*, pages 354–373. Springer, 2012.

[56] KyoungSoo Han, Jae Hyun Lim, and Eul Gyu Im. Malware analysis method using visualization of binary files. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, pages 317–321. 2013.

[57] Peter Harrington. *Machine learning in action*. Simon and Schuster, 2012.

[58] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[59] Sallie Henry. A technique for hiding proprietary details while providing sufficient information for researchers; or, do you recognize this well-known algorithm? *Journal of Systems and Software*, 8(1):3–11, 1988.

[60] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, 4:1039–1069, 2003.

[61] Pengfei Hu, Hongxing Li, Hao Fu, Derya Cansever, and Prasant Mohapatra. Dynamic defense strategy against advanced persistent threat with insiders. *IEEE Conference on Computer Communications*, pages 747–755, 2015.

[62] Julian Jang-Jaccard and Surya Nepal. A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, 80(5):973–993, 2014.

[63] Kangkook Jee, Vasileios P Kemerlis, Angelos D Keromytis, and Georgios Portokalidis. Shadowreplica: Efficient parallelization of dynamic data flow tracking. *ACM SIGSAC Conference on Computer & Communications Security*, pages 235–246, 2013.

[64] Yang Ji, Sangho Lee, Evan Downing, Weiren Wang, Mattia Fazzini, Taesoo Kim, Alessandro Orso, and Wenke Lee. RAIN: Refinable attack investigation with on-demand inter-process information flow tracking. *ACM SIGSAC Conference on Computer and Communications Security*, pages 377–390, 2017.

[65] Yang Ji, Sangho Lee, Evan Downing, Weiren Wang, Mattia Fazzini, Taesoo Kim, Alessandro Orso, and Wenke Lee. Rain: Refinable attack investigation with on-demand inter-process information flow tracking. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 377–390. ACM, 2017.

[66] Yang Ji, Sangho Lee, Mattia Fazzini, Joey Allen, Evan Downing, Taesoo Kim, Alessandro Orso, and Wenke Lee. Enabling refinable {Cross-Host} attack investigation with efficient data flow tagging and tracking. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1705–1722, 2018.

[67] Yang Ji, Sangho Lee, Mattia Fazzini, Joey Allen, Evan Downing, Taesoo Kim, Alessandro Orso, and Wenke Lee. Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking. *USENIX Security Symposium*, pages 1705–1722, 2018.

[68] Sami Kairajärvi, Andrei Costin, and Timo Hämäläinen. ISAdetect: Usable automated detection of CPU architecture and endianness for executable binary files and object code. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, pages 376–380, 2020.

[69] Aditya Kapoor. *An approach towards disassembly of malicious binary executables*. PhD thesis, University of Louisiana at Lafayette, 2004.

[70] Martin Karresand and Nahid Shahmehri. File type identification of data fragments by their binary structure. In *Proceedings of the IEEE Information Assurance Workshop*, pages 140–147, 2006.

[71] M Ammar Ben Khadra, Dominik Stoffel, and Wolfgang Kunz. Speculative disassembly of binary code. In *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*, pages 1–10. IEEE, 2016.

[72] J Zico Kolter and Marcus A Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7(12), 2006.

[73] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[74] Yonghwi Kwon, Dohyeong Kim, William Nick Sumner, Kyungtae Kim, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. Ldx: Causality inference by lightweight dual execution. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 503–515, 2016.

[75] Yonghwi Kwon, Fei Wang, Weihang Wang, Kyu Hyung Lee, Wen-Chuan Lee, Shiqing Ma, Xiangyu Zhang, Dongyan Xu, Somesh Jha, Gabriela F Ciocarlie, et al. Mci: Modeling-based causality inference in audit logging for attack investigation. In *NDSS*, volume 2, page 4, 2018.

[76] Phillip Lee, Andrew Clark, Basel Alomair, Linda Bushnell, and Radha Poovendran. A host takeover game model for competing malware. *IEEE Conference on Decision and Control*, pages 4523–4530, 2015.

[77] Yeo Reum Lee, BooJoong Kang, and Eul Gyu Im. Function matching-based binary-level software similarity calculation. In *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, pages 322–327. 2013.

[78] Bo Li, Phani Vadrevu, Kyu Hyung Lee, Roberto Perdisci, Jienan Liu, Babak Rahbarinia, Kang Li, and Manos Antonakakis. Jsgraph: Enabling reconstruction of web attacks via efficient tracking of live in-browser javascript executions. In *NDSS*, 2018.

[79] Jun Li. Learning average reward irreducible stochastic games: Analysis and applications. *Ph.D. dissertation, Dept. Ind. Manage. Syst. Eng., Univ. South Florida, Tampa, FL, USA*, 2003.

[80] Jun Li, Kandethody Ramachandran, and Tapas K Das. A reinforcement learning (nash-R) algorithm for average reward irreducible stochastic games. *Journal of Machine Learning Research*, 2007.

[81] Wenjia Li, Neha Bala, Aemun Ahmar, Fernanda Tovar, Arpit Battu, and Prachi Bambarkar. A robust malware detection approach for android system against adversarial example attacks. In *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*, pages 360–365. IEEE, 2019.

[82] Michael L Littman et al. Friend-or-foe Q-learning in general-sum games. In *ICML*, volume 1, pages 322–328, 2001.

[83] Kaiping Liu, Hee Beng Kuan Tan, and Xu Chen. Binary code analysis. *Computer*, 46(8):60–68, 2013.

[84] Kong-wei Lye and Jeannette M Wing. Game strategies in network security. *International Journal of Information Security*, 4(1-2):71–86, 2005.

[85] Yuan Ma, Lifang Han, Huan Ying, Shouguo Yang, Weiwei Zhao, and Zhiqiang Shi. Svm-based instruction set identification for grid device firmware. In *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, pages 214–218. IEEE, 2019.

[86] Carlos Martin and Tuomas Sandholm. Efficient exploration of zero-sum stochastic games. *arXiv preprint arXiv:2002.10524*, 2020.

[87] Mason McDaniel and Mohammad Hossain Heydari. Content based file type detection algorithms. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, pages 10–pp. IEEE, 2003.

[88] Timothy McIntosh, Paul Watters, ASM Kayes, Alex Ng, and Yi-Ping Phoebe Chen. Enforcing situation-aware access control to build malware-resilient file systems. *Future Generation Computer Systems*, 115:568–582, 2021.

[89] Jean-Francois Mertens and T Parthasarathy. Equilibria for discounted stochastic games. *Stochastic Games and Applications*, pages 131–172, 2003.

[90] Minghui Min, Liang Xiao, Caixia Xie, Mohammad Hajimirsadeghi, and Narayan B Mandayam. Defense against advanced persistent threats in dynamic cloud storage: A colonel blotto game approach. *arXiv preprint arXiv:1801.06270*, 2018.

[91] Shana Moothedath, Dinuka Sahabandu, Joey Allen, Andrew Clark, Linda Bushnell, Wenke Lee, and Radha Poovendran. A game-theoretic approach for dynamic information flow tracking to detect multistage advanced persistent threats. *IEEE Transactions on Automatic Control*, 65(12):5248–5263, 2020.

[92] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 421–430. IEEE, 2007.

[93] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[94] John F Nash. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[95] John F Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[96] Gleb Naumovich and Nasir Memon. Preventing piracy, reverse engineering, and tampering. *computer*, 36(7):64–71, 2003.

[97] James Newsome and Dawn Song. Dynamic taint analysis: Automatic detection, analysis, and signature generation of exploit attacks on commodity software. *Network and Distributed Systems Security Symposium*, 2005.

[98] Pietro De Nicolao, Marcello Pogliani, Mario Polino, Michele Carminati, Davide Quarta, and Stefano Zanero. ELISA: ELiciting ISA of raw binaries for fine-grained code and data separation. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 351–371. Springer, 2018.

[99] Edmund B Nightingale, Daniel Peek, Peter M Chen, and Jason Flinn. Parallelizing security checks on commodity hardware. *ACM Sigplan Notices*, 43(3):308–318, 2008.

[100] James B Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[101] Asuman Ozdaglar, Muhammed O Sayin, and Kaiqing Zhang. Independent learning in stochastic games. *arXiv preprint arXiv:2111.11743*, 2021.

[102] Christos H Papadimitriou and Tim Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM*, 55(3):14:2–29, 2008.

[103] Julien Pérolat, Florian Strub, Bilal Piot, and Olivier Pietquin. Learning nash equilibrium for general-sum markov games from batch data. In *Artificial Intelligence and Statistics*, pages 232–241. PMLR, 2017.

[104] HL Prasad, Prashanth LA, and Shalabh Bhatnagar. Two-timescale algorithms for learning Nash equilibria in general-sum stochastic games. *International Conference on Autonomous Agents and Multiagent Systems*, pages 1371–1379, 2015.

[105] Babak Bashari Rad, Maslin Masrom, and Suahimi Ibrahim. Opcodes histogram for classifying metamorphic portable executables malware. In *2012 International Conference on e-Learning and e-Technologies in Education (ICEEE)*, pages 209–213. IEEE, 2012.

[106] TES Raghavan and Jerzy A Filar. Algorithms for stochastic games—A survey. *Zeitschrift für Operations Research*, 35(6), 1991.

[107] Paweł Rajba and Wojciech Mazurczyk. Information hiding using minification. *IEEE Access*, 9:66436–66449, 2021.

[108] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[109] Stefan Rass, Sandra König, and Stefan Schauer. Defending against advanced persistent threats using game-theory. *PLoS one*, 12(1):e0168675: 1–43, 2017.

[110] ReFirmLabs. Refirmlabs/binwalk: Firmware analysis tool.

[111] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[112] D. Sahabandu, Sukarno Mertoguno, and Radha Poovendran. A natural language processing approach for instruction set architecture identification. *Accepted to IEEE Transactions on Information Forensics and Security*, 2023.

[113] Dinuka Sahabandu, Shana Moothedath, Joey Allen, Linda Bushnell, Wenke Lee, and Radha Poovendran. Stochastic dynamic information flow tracking game with reinforcement learning. In *International Conference on Decision and Game Theory for Security*, pages 417–438. Springer, 2019.

[114] Dinuka Sahabandu, Shana Moothedath, Joey Allen, Andrew Clark, Linda Bushnell, Wenke Lee, and Radha Poovendran. Dynamic information flow tracking games for simultaneous detection of multiple attackers. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 567–574. IEEE, 2019.

[115] Dinuka Sahabandu, Shana Moothedath, Linda Bushnell, Radha Poovendran, Joey Aller, Wenke Lee, and Andrew Clark. A game theoretic approach for dynamic information flow tracking with conditional branching. In *2019 American Control Conference (ACC)*, pages 2289–2296. IEEE, 2019.

[116] Dinuka Sahabandu, Baicen Xiao, Andrew Clark, Sangho Lee, Wenke Lee, and Radha Poovendran. Dift games: Dynamic information flow tracking games for advanced persistent threats. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 1136–1143. IEEE, 2018.

[117] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th international conference on malicious and unwanted software (MALWARE)*, pages 11–20. IEEE, 2015.

[118] Muhammed Sayin, Kaiqing Zhang, David Leslie, Tamer Basar, and Asuman Ozdaglar. Decentralized Q-learning in zero-sum markov games. *Advances in Neural Information Processing Systems*, 34, 2021.

[119] Matthew G Schultz, Eleazar Eskin, F Zadok, and Salvatore J Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pages 38–49. IEEE, 2000.

[120] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). *IEEE Symposium on Security and Privacy*, pages 317–331, 2010.

[121] Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[122] Karl A Sickendick. File carving and malware identification algorithms applied to firmware reverse engineering. 2013.

[123] Matthew Sobel. Noncooperative stochastic games. *The Annals of Mathematical Statistics*, 42(6):1930–1935, 1971.

[124] G Edward Suh, Jae W Lee, David Zhang, and Srinivas Devadas. Secure program execution via dynamic information flow tracking. *ACM SIGPLAN Notices*, 39(11):85–96, 2004.

[125] Pengfei Sun, Luis Garcia, Gabriel Salles-Loustau, and Saman Zonouz. Hybrid firmware analysis for known mobile and iot security vulnerabilities. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 373–384. IEEE, 2020.

[126] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[127] Milind Tambe. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press, 2011.

[128] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Evaluating adversarial attacks against multiple fact verification systems. Association for Computational Linguistics, 2019.

[129] Trung Kien Tran and Hiroshi Sato. Nlp-based approaches for malware classification from api sequences. In *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*, pages 101–105. IEEE, 2017.

[130] Bruno Trstenjak, Sasa Mikac, and Dzenana Donko. Knn with tf-idf based framework for text categorization. *Procedia Engineering*, 69:1356–1364, 2014.

[131] Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. Advanced persistent threats: Behind the scenes. *Annual Conference on Information Science and Systems (CISS)*, pages 181–186, 2016.

[132] Marten Van Dijk, Ari Juels, Alina Oprea, and Ronald L Rivest. Flipit: The game of "stealthy takeover". *Journal of Cryptology*, 26(4):655–713, 2013.

[133] Nicolas Vieille. Two-player stochastic games ii: The case of recursive games. *Israel Journal of Mathematics*, 119(1):93–126, 2000.

[134] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. *Network & Distributed System Security Symposium*, 2007.

[135] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, 2002.

[136] Ke Wang, Janak J Parekh, and Salvatore J Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection: 9th International Symposium, RAID 2006 Hamburg, Germany, September 20-22, 2006 Proceedings 9*, pages 226–248. Springer, 2006.

[137] Bryan Watkins. The impact of cyber attacks on the private sector. pages 1–11, 2014.

[138] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5(2):32–39, 2007.

[139] Hongfa Xue, Shaowen Sun, Guru Venkataramani, and Tian Lan. Machine learning-based analysis of program binaries: A comprehensive study. *IEEE Access*, 7:65889–65912, 2019.

[140] Tarun Yadav and Arvind Mallari Rao. Technical aspects of cyber kill chain. *International Symposium on Security in Computing and Communication*, pages 438–452, 2015.

[141] Abbas Yazdinejad, Hamed HaddadPajouh, Ali Dehghantanha, Reza M Parizi, Gautam Srivastava, and Mu-Yen Chen. Cryptocurrency malware hunting: A deep recurrent neural network approach. *Applied Soft Computing*, 96:106630, 2020.

[142] Hongli Yuan, Yongchuan Tang, Wenjuan Sun, and Li Liu. A detection method for android application security based on tf-idf and machine learning. *Plos one*, 15(9):e0238694, 2020.

[143] Hanqi Zhang, Xi Xiao, Francesco Mercaldo, Shiguang Ni, Fabio Martinelli, and Arun Kumar Sangaiah. Classification of ransomware families with machine learning based onn-gram of opcodes. *Future Generation Computer Systems*, 90:211–221, 2019.

[144] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control*, pages 321–384, 2021.

[145] Quanyan Zhu and Tamer Başar. Robust and resilient control design for cyber-physical systems with an application to power systems. *IEEE Decision and Control and European Control Conference (CDC-ECC)*, pages 4066–4071, 2011.