THE UNIVERSITY OF WINNIPEG

MASTER'S THESIS

Securing Intrusion Detection Systems in IoT Networks Against Adversarial Learning: A Moving Target Defense Approach based on Reinforcement Learning

Author:
Arnold Brendan Osei

Supervisors:
Dr. Talal HALABI
Dr. Yaser AL MTAWA

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science
in the
Department of Applied Computer Science

August 26, 2023

Declaration of Authorship

“*Like a simple symphony, no we aim not for ending chords. The motive for the play is just to keep enjoying it.*”

Arnold Brendan Osei

THE UNIVERSITY OF WINNIPEG

Abstract

Faculty of Science

Department of Applied Computer Science

Master of Science

Securing Intrusion Detection Systems in IoT Networks Against Adversarial Learning: A Moving Target Defense Approach based on Reinforcement Learning

by Arnold Brendan Osei

Investigating the use of moving target defense (MTD) mechanisms in IoT networks is ongoing research, with unfathomable potential to equip IoT devices and networks with the ability to fend off cyber attacks despite the computational deficiencies many IoT ecosystems typically have. The AI community has extensively studied adversarial threats and attacks on machine learning-based systems, emphasizing the need to address the potential compromise of anomaly-based intrusion detection systems (IDS) through adversarial attacks. Another concept that has gained significant attention in the networking community is Game Theory. Protecting any given network is almost a never-ending battle between the attacker and defender, and hence a natural game of competitors can be modelled based on one's parametric specifications to gain more insight into how attackers might interact with one's system. The goal of this thesis is to propose a comprehensive, experimentally verifiable game-theoretic model of MTD in IoT networks to secure the IDS against adversarial attacks. Once a game with state transitions based on given actions can be modelled, reinforcement learning is used to develop policies based on various episodes (rounds) of the game, ultimately optimizing network decisions to minimize successful attacks on machine learning-based IDS. The state-of-the-art ToN-IoT dataset was investigated for MTD feasibility to implement the feature-based MTD approach. The overall performance of the proposed MTD-based IDS was compared to a conventional IDS by analyzing the accuracy curve of the MTD-based IDS and the conventional IDS for varying attacker success rates and resource demands. Our approach has proven effective in securing the IDS against adversarial learning.

Acknowledgements

I would like to thank my supervisors Dr. Talal Halabi and Dr. Yaser Al Mtawa for the tremendous help and guidance throughout this research journey. A big thanks to my family and friends for always being there through thick and thin.

Contents

List of Figures

List of Tables

List of Abbreviations

IDS	**I**ntrusion **D**etection **S**ystems
IoT	**I**nternet of **T**hings
MTD	**M**oving **T**arget **D**efense
RL	**R**einforcement **L**earning
ML	**M**achine **L**earning
PCA	**P**rincipal **C**omponent **A**nalysis
IMD	**I**mplantable **M**edical **D**evices

To all Street Angels…

Chapter 1

Introduction

1.1 Motivation and Objectives

With the advent of exponentially growing technological mechanisms that have expanded communication bandwidths and made it possible for seamlessly ultra-fast inter-connectivity of inestimable nodes, there has been a global consensus to proactively evolve the internet to include not just human-to-human interactions through devices, but also device-to-human and device-to-device interactions, where these devices have "consciences" of their own and exist as independent entities in networks - Hence the global outburst of Internet of Things (IoT). A major concern about IoT systems however is security and privacy, which become all the more difficult to implement in these devices that are mostly embedded and have limited computational resources to run firewalls and advanced security algorithms. To be able to prevent IoT systems from being attacked, there ought to be more rigorous mechanisms that go beyond a typical firewall's inspection of headers of packets based on assigned rules, and rather try to extensively examine payloads of packets for potential anomalies – intrusion detection systems (IDS).

Intrusion detection systems (IDS's) can either be signature-based or anomaly-based. Signature-based IDS seemed to have been the natural way to go in the past, as known attacks were studied and their features examined and implemented as signatures on the IDS, so that their traffic could easily be identified by the IDS based on the known characteristics. However, cyber attackers device new ways of orchestrating attacks every now and then that are usually cunningly evasive, and could render signature-based detection defunct. This is why there have been growing interests and active research in anomaly-based intrusion detection systems; and with the onset of advanced machine learning techniques and algorithms, this proves to be the ultimate way to stop all forms of attacks (whether known or unknown), with the ultimate goal of being able to accurately distinguish normal traffic from malicious traffic, however difficult.

The goal of this thesis is to proactively protect an anomaly-based IDS from potential adversarial learning attacks using a novel Moving Target Defense (MTD) approach. Network traffic can be distilled into network datasets for the sake of analysis and training of anomaly-based IDS. As part of this research, the ToN-IoT dataset was rigorously studied and explored in diverse ways to ascertain if it can be used to achieve the objectives as stipulated. As would be determined later in this thesis, all theoretical ideas and corresponding experiments have been inspired by analyzing the ToN-IoT dataset.

This thesis focuses on establishing a theoretic model for an MTD-enabled IDS to prevent against adversarial threats. An adversarial attack is a kind of an attack that stealthily injects or places an input to a machine learning (ML) model that is purposely designed to cause a model to make a mistake in its predictions despite resembling a valid input when observed. There are three ways of looking at adversarial attacks: In white box adversarial attacks, the adversary completely knows the network. This would also imply a complete knowledge on the training and testing datasets of a given IDS, what ML model and miscellaneous techniques are employed by the IDS, among other things. In gray box adversarial attacks, the adversary (attacker) only has partial knowledge of the network. Lastly, black box adversarial attacks require that the adversary (attacker) has no knowledge of the network, and launches the attack blindly and arbitrarily.

1.2 Problem Definition

Conventional IDSs used as gateways in IoT networks are as susceptible to attacks as the nodes within these networks. Little attention has been placed on this, but the damage caused by a cyber attack that compromises an IDS could be far more detrimental than that caused by compromising nodes within the network, since the IDS's mission is to protect the network. Although researchers using MTD have made important strides within the community, little attention has been placed on implementing a proactive approach like MTD on IoT gateways involving IDS components.

This thesis attempts to address that by proposing an IDS architecture that lends itself naturally for the implementation of MTD, by leveraging the power of game theory and reinforcement learning.

1.3 Summary of Contributions

The contributions of this thesis are as follows:

- A novel MTD model based on decentralization and learning diversification to protect the IDS against adversarial threats.
- A reinforcement learning solution to autonomously optimize MTD deployment within the network.
- An extensive experimental validation of our proposed threat prevention solution using the state-of-the-art, real-world ToN-IoT dataset.

1.4 Thesis Layout

This thesis is structured as follows:

- Chapter 2 provides background information on the areas explored in this research.
- Chapter 3 provides a literature review on all related works to this thesis.
- Chapter 4 provides a detailed outline of the proposed MTD model and solution.
- Chapter 5 provides rigorous analysis of the ToN-IoT dataset, and investigates its viability for MTD implementation.
- Chapter 6 describes the experiments and results by evaluating the performance of the proposed solution compared to the conventional IDS in IoT networks.
- Chapter 7 highlights the limitations and conclusions to be drawn from the outcomes of this research, and outlines the promising future works that this research opens the door to.

Chapter 2

Background Information

2.1 Security Threats in IoT Networks

Internet of Things (IoT) has become an intrinsic technology in various automated industries and large-scale smart city and government services including wearable health devices and autonomous transportation Makhdoom et al., 2018. IoT involves sensors that transmit data to the cloud and control the decisions of cyber-physical processes. According to recent statistics IHS, 2018, there are currently over 26 billion active IoT connected devices worldwide.

Computer networks in general are usually looked at in a systematic manner, through layers. This layer partitioning approach facilitates comprehensive and standard network design layouts and analysis. To understand the threats posed to IoT networks, it is important to understand generic IoT network layers, as usually, threats launched on IoT networks could be identified in the context of the specific layer their design and use-case is based off. IoT consists of four main layers: the physical layer that includes IoT sensors and actuators; the network layer (edge-cloud communications) responsible for transferring the data from devices for processing in upper layers; the processing layer that leverages the cloud environment to perform computational tasks; and the application layer delivered via end-user devices. All layers are subject to security threats including Denial of Service (DoS) attacks Giraldo et al., 2018; Rubio, Alcaraz, and Lopez, 2017.

Jamming is an IoT security threat that compromises the perception layer of an IoT network. The attacker craftily jams media access channels of IoT nodes, thereby creating problems for legitimate node connectivity within the network Zaman et al., 2021. Jamming threats can be done in various forms: injecting continuous random wave forms into the channel would constitute a constant jammer; listening to activity within the channel before injecting noise signals in the channel constitutes a reactive jammer threat; whereas a strategic jammer would consist of a more intelligent optimal approach for injecting jamming signals within a given IoT channel Gwon et al., 2013.

MAC address spoofing is an attack within the perception layer of an IoT network. The deliberate attempt to change the MAC address of a given IoT node to cause identity problems in node-to-node communications and node-to-gateway communication is MAC spoofing. Bander et al Alotaibi and Elleithy, 2016 developed a novel passive technique that detects MAC address spoofing in wireless sensor nodes based on a mainstream machine learning classification algorithm, namely the random forest ensemble method.

Attacks that target how IoT nodes conserve energy and resources, for example by tweaking the sleep time of certain wireless sensor nodes so that they expend more resources than needed, are among the most adverse perception layer security threats for IoT networks. Hei et al Hei et al., 2010 conducted analysis on resource depletion attacks in the perception layer by using a simulation tool known as a software radio in implantable medical devices (IMDs). It was proven that an arbitrary attacker posed a serious threat on the life of patients just by reducing the battery span of an IMD .

The network layer is the prime focus as far as the architecture for node-to-node communication and node-to-server/gateway communication is concerned. It also involves numerous protocols like IPv4 and IPv6 that could be easily exploited. Consequently, it is liable to all forms of attacks including distibuted DoS (DDoS), DoS, spoofing attacks, eavesdropping, and man-in-the-middle attacks.

IoT devices and networks are known to be increasingly vulnerable to security attacks on data integrity and service availability. Security threats constitute a major obstacle against the achievement of reliable IoT systems. Indeed, current security vulnerabilities in IoT devices have led to several cases of DoS attacks Antonakakis et al., 2017. The attacks launched on the processing layer such as data corruption attacks (e.g., data inconsistency, unauthorized access), malware, and DDoS attacks could also affect the cloud server and delivered services. With increased connectivity, IoT devices can become compromised and used as zombies. Hackers can control these devices remotely and utilize them for illegal purposes and carry out large-scale DDoS attacks, e.g., via a Control & Command (C&C) server Şendroiu and Diaconescu, 2018.

2.2 Intrusion Detection Systems in IoT

Intrusion detection systems can be either host-based (HIDS) or network-based (NIDS). The host-based IDSs are attached to the devices/nodes themselves to shield them from attacks, whereas the network-based IDS monitor the entire IoT network's traffic Santos, Rabadao, and Gonçalves, 2018. As most IoT devices are deficient in computational and memory resources, most IoT networks rely on network-based IDSs to provide security for the collective nodes within the network Elrawy, Awad, and Hamed, 2018. Hence moving forward in this thesis, the mention of intrusion detection systems refers to network-based intrusion detection systems (NIDS). On the other hand, IDSs can either be signature-based or anomaly-based. Signature-based IDSs seemed to have been the natural way to go in the past, as known attacks were studied and their features examined and implemented as signatures on IDSs so that their traffic could easily be identified based on the known characteristics. However, cyber attackers device new ways of orchestrating attacks every now and then that are usually cunningly evasive, and could render signature-based detection defunct. This is why there have been growing interests and active research in anomaly-based intrusion detection systems; and with the onset of advanced machine learning techniques and algorithms, this proves to be the ultimate way to stop all forms of attacks (whether known or unknown), with the ultimate goal of being able to accurately distinguish normal traffic from malicious traffic, however difficult. Naturally, when this thesis talks about IDS, it is referring to anomaly-based IDS as these are the future for detecting zero-day and unknown attacks. It is expected that state-of-the-art IoT networks implement anomaly-based IDS, and any signature-based functionalities may be included as add-ons. The accuracy of anomaly-based IDS's hinges on some questions: What is the most appropriate ML model to use? What data sets and classification features should be used to train and test this model?

Assuming that the most ideal machine learning model is chosen, the accuracy of an anomaly-based IDS would greatly depend on what kind of data sets are used to train the model it uses, hence the motivation for the analysis of the various datasets out there in the literature leading up to the adoption of the ToN-IoT dataset for this projec Alsoufi et al., 2021.

2.3 Anomaly Detection Datasets

Current research to come up with optimal datasets for IoT intrusion detection systems has been somewhat lacking. Currently available data sets are, in one way or another, derivatives of the benchmark NSL-KDD dataset, which has become more or less the standard for generic training of anomaly-based network IDS. This dataset itself is a revised version of the KDD-CUP'99 data set (prepared by Stolfo et al., 2000), which was also inspired by data records obtained from the 1998 DARPA program at the MIT Lincoln Labs. The authors in Tavallaee et al., 2009 proposed the NSL-KDD dataset as the ultimate successor of the KDDCUP'99 (hereby shortened to "KDD'99") by meticulously analyzing the KDD'99 dataset for possible flaws.
The authors in Hindy et al., 2020 came up with an IoT-specific dataset that was generated by simulating MQTT-based traffic. Their experiment was conducted by attempting to simulate an MQTT machine-to-machine network architecture of 12 sensors, a broker, a camera, and an attacker. Also, five scenarios of data records were recorded: Normal operation, aggressive scan, UDP scan, Sparta SSH brute-force attack, and MQTT brute-force attack. Three abstraction levels of features are also employed in the composition of this dataset (after extraction from raw pcap files): packet features, unidirectional flow features and bidirectional flow features. The highlight of this dataset is perhaps that although it is IoT-specific, the traffic was synthetically generated and may not be reflective of real-world scenarios. It seems to share most of the loopholes expressed in the aforementioned datasets, and confidence in this dataset is yet to be fully ascertained by the IoT security community. From the above mentioned, it can be clearly seen that the research extent around the development of comprehensive datasets for IoT anomaly-based intrusion detection systems is still in its infancy.

2.4 Adversarial Learning Threats to IDS

An adversarial learning attack is a kind of attack that stealthily injects or places an input to a machine learning model, purposely designed to cause a model to make a mistake in its predictions despite resembling a valid input when observed. There are typically three ways of looking at adversarial attacks: White box adversarial attacks - This is when the adversary completely knows the network. That would also imply a complete knowledge on the training and testing datasets of a given IDS, what ML model and miscellaneous techniques are employed by the IDS, etc. In Gray box adversarial attacks the adversary (attacker) only has partial knowledge of the network. Finally, in black box adversarial attacks the adversary (attacker has no knowledge of the network and launches the attack blindly and arbitrarily.

The approaches to executing an adversarial learning attack can be grouped into reactive and proactive approaches:

- Reactive Approach: Involves carrying out patches on a given network, based on the kind of adversarial attacks experienced. This could be likened to adapting the IDS as a sort of signature-based IDS. This approach could be done in an iterative manner to enhance the robustness of the IDS.
- Proactive Approach: Involves altering the underlying architecture or learning procedure of the IDS by, for example, adding more layers, training in real time the IDS with adversarial attack samples, or increasing the sensitivity of loss/activation functions . The loss function is given by:
  $f(x+\delta) = f(x)$, Where δ is the input injected by the adversary to cause perturbation and impact prediction accuracy of the IDS.

Fast Gradient Sign Method (FGSM) is a kind of adversarial learning attack that uses the concept of gradient descent in neural networks, which is in turn an iterative optimization process to minimize the adversarial error during an attempt to compromise a given deep learning model Huang et al., 2017. This results in data points that look indistinguishable from the original ones but results in serious misclassification when passed through a deep learning model. Adversarial training too could be used, where the IDS is trained with some adversarial examples so as to make it better immune to adversarial attacks. The Barrage of Natural Transforms (BaRT) randomly sets up the classifier to be vulnerable to a number of transforms, based on the criteria which the classifier/ML model uses in its prediction. Some transforms could be: Noise injection, FFT perturbation and Color precision reduction (in the case of image classifications) Mahmood et al., 2021. In a jacobian based saliency attack (JSMA), by analyzing the jacobian matrix of outputs with respect to inputs, one is able to deduce how the output probabilities behave given a slight modification of an input feature. In Ayub et al., 2020, JSMA was used against a multilayer perceptron (MLP) model by using the CIDS and TRAbID datasets for network traffic classification. All these adversarial threats can render the IoT network highly vulnerable to compromise by subverting the performance of the IDS.

Chapter 3

Literature Review

3.1 Moving Target Defense (MTD)

3.1.1 MTD in IoT Systems

A standard traditional network could be set up to have the best security mechanisms implemented, with state-of-the-art firewalls, IDS, cryptographic techniques, etc. The simple fact is if some arbitrary attacker has an interest in breaking through the system, and has adequate resources at his/her disposal, with enough time, any static network arrangement can be breached and compromised. Perhaps there could be an argument made that even if the attacker is successful at an attack launch, these vulnerabilities could be patched to further reduce the attack surface available for any prospective attacks. However, how further down can an attack surface be shrunk? How about zero-day vulnerabilities that have not yet been detected, talk less of being patched? Also, these standard traditional ways of dealing with compromised systems are highly reactive. It means the damage would have been caused before counteraction is initiated, and there are negative impacts on network operations due to constant patches. It is also easier to create backdoors in traditional systems if system parameters are unchanging/static. Furthermore, resource-constrained devices and networks like IoT systems may not be able to add complex security set-ups and hence are most vulnerable to attacks. These and perhaps many more have been motivating factors for shifting the focus from traditional, static networks to more dynamic set-ups based on the Moving Target Defense paradigm (MTD).

MTD is a cyber-defence paradigm that proposes to proactively make systems and networks dynamic, so as to make it even more difficult for an attacker to be successful with exploits. With MTDs, there is a full-on acknowledgement that systems are always going to be vulnerable regardless of how many times attack surfaces are shrunk, because there is an unfair asymmetrical advantage that attackers have on static systems over time, hence reconnaissance activities of the attacker on a static network would render the attacker in a favorable position to be successful with an attack Navas et al., 2020.

As can be seen in figure 3.1, MTD research is typically focused on the nodes, and is oblivious to the gateways through which all traffic converge and diverge. The gateway being referred to in this case is the intrusion detection system (IDS). In principle, no matter how safe we make IoT nodes by virtue of proactive approaches such as MTD, if gateways including the IDS are vulnerable to compromise, the entire network has a fundamental security loophole that can be exploited.

To counteract this asymmetrical advantage that an attacker has over a traditional network, MTD proposes the movement of system parameters at certain periods so that the state of the network at T_0 is different from the state of the network at some arbitrary period $T_0 + \delta t$, thereby making it difficult for the attacker to do any proper reconnaissance and launch a successful attack. The goal in this case of MTD is not to reactively reduce the attack surface with traditional countermeasures like patches, but rather to proactively keep moving the attack surface to make it seemingly impossible or very difficult to be successful with an attack, as can be seen from figure 3.1. MTDs have to do with movements, so one key design step to take in setting up an MTD-based network is to know what to move, how to move it, and when to move it, as can seen from figure 3.2.

With regards to what parameters to move/change, this could be data (changing data formats, etc), software, network (changing IP addresses, port numbers, etc), platform (OS changes, firmware changes), runtime environment (RAM address space changes, etc), or perhaps even hardware (routine changes in enterprise switch brands, etc). With regards to how to move, the moving parameter (MP) can be made to move from one state to another via shuffling (randomization) or perhaps using some predefined optimization algorithm, etc. With regards to when to move (the trigger to initiate the move), this could be done based on a specific time or event, or a hybrid combination of both. Because of the uncertainty created by changing configurations on the network, it is possible to quantitatively attempt to describe the degree of uncertainty of a given network with say a chosen moving parameter (MP) by evaluating the number of states the MP is capable of taking on, and the probability that it takes on a certain state. This can be modeled using Shannon's entropy.

For a uniform probability distribution, Shannon's entropy will directly depend on the number of states available for a given MP Zhuang, DeLoach, and Ou, 2014. This theoretical inclination further buttresses the point that: Instead of reducing the attack surface, MTDs rather enlarge the "exploration surface" domain for attackers, and then moves the attack surface as a sub-domain with the exploration domain. Thereby making it difficult for attackers to orchestrate attacks. That said, there is always going to be an element of qualitative measurement when MTD is concerned. This is because even though more states of an MP translate to higher uncertainty and hence greater difficulty for the attacker, it could be much more difficult and costlier to have many states, depending on the moving parameter in question. For example, it is easier to have multiple states of IP addresses compared to say multiple firmware. This also means for an attacker, it may be easier to break through say 254 different states of a node's IP address than 5 different states of a node's firmware. Hence there ought to be some qualitative representation of weighing the cost of states of a given MP, for both the attacker and the defender.

In a given network environment, a typical use-case could be random re-assignment of IP addresses and port numbers. This can thwart the reconnaissance activities of the attacker. For example, using scanning tools like nmap will yield different results for each scan and will therefore not be useful knowledge to the attacker. Another use-case could be constant changing of communication protocols between nodes and gateway. For example in IoT networks, nodes could communicate to the gateway using protocols like WiFi, Bluetooth, Zigbee, etc. Constantly changing communication protocol would make it difficult for an attacker to launch an attack as each protocol is completely different from the other, with no correlation whatsoever. The moving target defense strategy is a relatively new area of research that is gaining momentum, especially for low-resource networks like IoT networks Mercado-Velázquez, Escamilla-Ambrosio, and Ortiz-Rodriguez, 2021.

The authors in Jia, Sun, and Stavrou, 2013 implement a moving target defense model: MOTAG, that invlolves shuffling proxies of which clients ought to connect to access system resources: The prime objective of their strategy is to dissociate insider attackers from true clients. This is done by categorizing the proxies to which clients are connected to as either serving proxies or shuffling proxies during a DDoS attack. The serving proxies contain clients that do not experience any attack impacts, while the shuffling proxies are for the clients' experiencing attacks. The idea is to keep shuffling (randomly re-distributing) clients among the shuffling proxies for each iterative shuffle process, and adding proxies that do not experience attacks to the “serving proxies” group. This is continuously done until such a time as the clients responsible for the insider attacks are completely isolated. There is also an optimization goal to maximize the number of clients free from attacks after each iterative shuffle, using optimization techniques, and making sure there are enough shuffle proxies available to isolate insider attacks as fast as possible.

The main objective of the MOTAG architecture is to protect system assets (application server in this case) from attacks, especially DDoS and flooding attacks. It does this by introducing an intermediate layer made up of proxies, to diffuse and/or absorb attack traffic, and hence shield the application server (asset to be protected). The main idea is to provide a first layer of security to the intermediate layer (which is the point of contact with clients), so that even if an attacker is successful in compromising a proxy, it has another layer of security that governs communication between proxies and the server.

The authors in Wang and Wu, 2016 introduce a novel moving target defense against network reconnaissance. They call this MTD-based software-defined technique the Sniffer Reflector. This MTD architecture is basically set up to prevent successful network probing by the attacker by providing forged responses to network scans. The attacker gets feedback for each network scan conducted, however the scans targeting the main network are reflected on to a shadow network, which simulates scan replies that are sent back to the attacker. The attacker hence assumes these are legitimate responses from the targeted network, while these have been obfuscated by a shadow network that mimics the original network, thereby rendering the attacker’s reconnaissance invalid.

The authors in Giraldo, El Hariri, and Parvania, 2022 propose a novel MTD framework employing IoT-enabled data replication to replicate sensory and control signals in cyber-physical systems. This framework combines two layers of uncertainty, hence reducing the arbitrary attacker’s ability to learn about the IoT network over time. It also reduces the impact of false data injection attacks on a given system model.

3.1.2 Game Theory for MTD

Game Theory has been largely used to evaluate situations where individuals and organizations can have conflicting objectives. A game may be defined as a strategic interaction between two or more entities, where the entities involved act in such a manner as to maximize

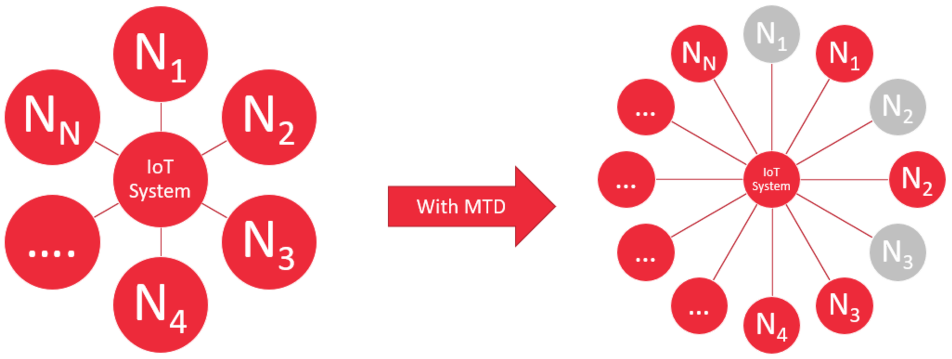


FIGURE 3.1: Overview of the MTD process applied to IoT networks.

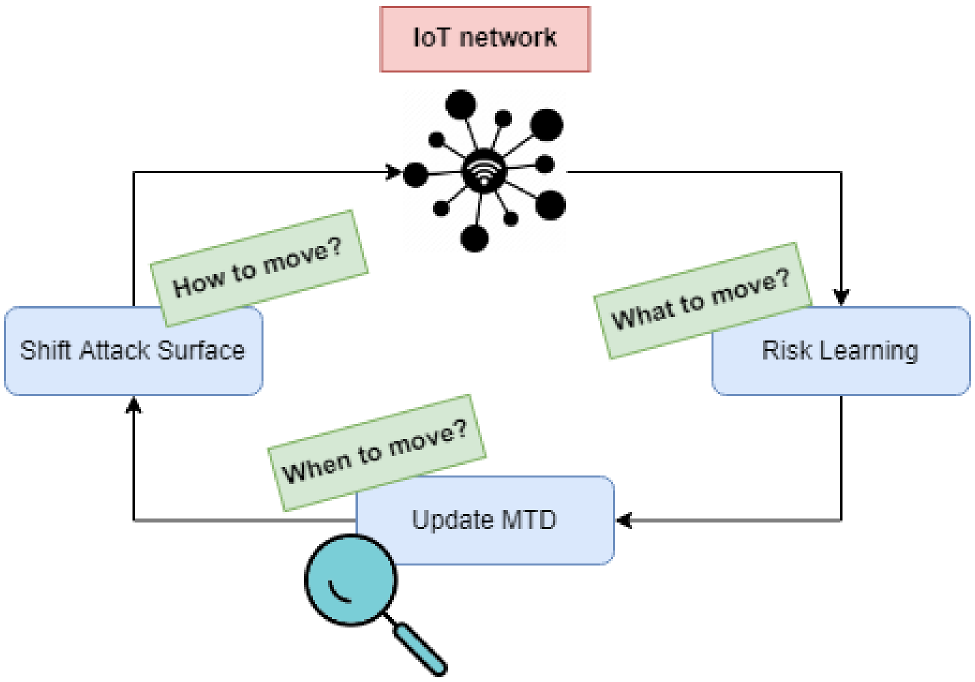


FIGURE 3.2: Overview of the MTD process applied to IoT networks.

their wins and minimize their losses (whether cooperatively or competitively) Von Neumann and Morgenstern, 2007. The Central decision makers in any given game that either work cooperatively or against each other to maximize their profits and minimize their losses are known as players, while the series of moves that each player in the game can make, where each action has direct consequences on the utilities of players are termed as actions. Pay-offs or Utilities are the quantifiable motivations that players get for executing corresponding actions. They serve to motivate/ de-motivate a player from playing a given action/actions. If a game involves only two players (competitors), then it is called a two-person game. If more than two players are involved, then the game is referred to as n-person game. For a two-player game, if the reward obtained from the utility function of one player is equal in magnitude but opposite in sign to the utility value obtained by the other player such that the sum of their utility values equal zero, the game is termed as a zero-sum game Maschler, Zamir, and Solan, 2020.

An MTD system typically requires continuous adaptations of system configurations to be able to effectively hinder attacks. This is expected to cause some overhead on the resources of a given system. On the other hand, limiting adaptations in a bid to reduce the overhead could also make it much easier for attackers to successfully execute attacks. Therefore, determining the right time to make adaptations, with the objective of minimizing long-term costs, highlights one of the keen problems in MTD - The MTD timing problem. The authors in Wang, Li, and Chen, 2016 propose a model to deal with the timing problem by coming up with an intelligent and optimal way to make adaptations (smart and optimized adaptations), which would naturally also include a trade-off between reducing system resource overhead and increasing the resilience of the system to attacks. This was done using an MTD framework that comprises of: defender system, adaptation cost analysis block, attacked cost analysis block, distribution fitting block, and adaptation analysis engine block. The defender system is supposed to make economical and optimized adaptations based on a loopback communication with the adaptation analysis engine block. Before this happens, the system feeds various information parameters to the other blocks, which in turn do some computational analysis, and then all the data gets aggregated at the adaptation analysis engine block, which makes sense of the data and guides the defender system to make an adaptation. The adaptation cost analysis block computes the adaptation cost resulting in the movement from one configuration state to another. The attack cost analysis block computes the cost incurred by the system after a successful attack. The distribution fitting block computes the distribution pattern and intervals of historical attacks on the system.

The authors in Osei et al., 2022 proposed a game-theoretic MTD model to address the MTD timing problem. In their model, they investigated two DDoS attacks: ICMP flooding and SYN Spoofing attacks. They theoretically calculated the time taken for an arbitrary attacker to successfully launch either of these attacks, provided certain parameters like the byte size and number of requests for a TCP connections table were given. Based on these calculations they formulated a game where the defending system had four options: To shuffle IoT nodes at a time t less than the time taken to successfully launch an ICMP flooding attack (which took less time to launch than the SYN spoofing attack); To shuffle at a time t in between these two attack success times; To shuffle nodes at a time greater than the SYN spoofing attack time; and then to shuffle nodes at $t = \infty$. The attacker on the other hand has two actions: To either launch an ICMP flooding attack or a SYN spoofing attack. Each player's utilities depended on what action was being taken, the number of shuffled nodes, the number of compromised nodes, and the total number of nodes within the network. The game was played between both players until Nash equilibrium (the equilibrium point of optimality of either parties, where either player's move was independent of the other). The utilities of each player was plotted for purely random moves (blind adaptations), and then for smart adaptations (moves that were made based on the probabilities reached at Nash Equilibrium). They were able to prove that their game-theoretic model provided some guided framework to ensure that the defending system moved at on optimal time to yield the most rewards and shield the network against attacks.

3.2 Defense Against Adversarial Attacks

One of the ways to defend against adversarial attacks is to train the IDS with some adversarial examples during the training phase Xu et al., 2020. This can sometimes lead to label leakage and over-fitting as the adversarial examples generated during the training phase may not be present during the predictive/testing phase. The authors in Zhang and Wang, 2019 came up with a novel model for adversarial training that involves feature scattering in a given latent space. They generate the feature-scattering adversarial examples in an unsupervised manner as a deliberate attempt to address possible label leakage. They analyse their proposed model and test it on various datasets to prove its efficacy. Fast gradient sign method (FGSM) is one of the most common adversarial learning attacks in the literature and is mostly used in image misclassification. To defend against this kind of attack, gradient masking (that naturally transforms a threat model from a white/gray box into a black box) is used to mask the model's output with respect to its input Lee, Bae, and Yoon, 2020.

The authors in Zantedeschi, Nicolae, and Rawat, 2017 conducted experiments on two standards datasets: the MNIST and the CIFAR10. They generated adversarial examples using adversarial attacks such as the fast gradient sign method (after a preliminary step of adding gaussian noise), the jacobian saliency map method, the deepfool method with 100 iterations, and the L_2 method from Carlini and Wagner (C& W) Carlini and Wagner, 2017. They further compare the effectiveness of adversarial defense methods such as feature squeezing (FS), label smoothing (LS), adversarial training (adversarial methods crafted using the FGSM attack method), and gaussian data augmentation (generating ten noisy samples for each original one, and a standard deviation of 0.3 for MNIST and 0.05 for CIFAR10).

A novel GAN-based adversarial defense method called Cowboy was proposed by Santhanam and Grnarova, 2018. This approach both detects and defends against adversarial attacks by using both the descriminator and generator of a typical GAN trained on the same dataset. They analyze the performance of this proposed approach using the MNIST and CIFAR-10 datasets that already exist in the literature. The method is inspired by hypothesizing that adversarial samples ought to exist out of the data pipeline understudied by a generative adversarial network.

3.3 Reinforcement Learning for Cyber Defense

Reinforcement learning is a machine learning concept in which the agent learns about its environment by trial-and-error. The agent undertakes a sequence of actions that yield responses or feedback signals from the environment in the form of rewards or punishments. Ultimately, the agent learns over time based on the actions that were taken and the rewards/punishments that were obtained over time. The environment the agent plays in evolves over time based on the actions of the agent. The environment state transitions can be seen as stochastic sequences or Markov decision processes Li, 2017. Each termination point of reinforcement learning setup points to the end of an episode, and so the whole premise of reinforcement learning could be seen through the lens of a stochastic game being played by the agent, where the game terminates in various ways (multiple episodes), and the agent's ultimate task is to find an optimal policy that yields the most rewards, after experiencing several episodes of the game Ding et al., 2018.

The authors in Feng and Xu, 2017 developed an optimal online defense strategy for cyber-physical systems given an arbitrary attack. The problem was formulated as a zero-sum game of two players, and the game-theoretic neural network structure was proposed to learn the optimal defense strategy online. To enhance the proposed scheme in real time, a deep reinforcement learning algorithm was formulated to facilitate the fine-tuning of the neural network. Their simulation results revealed that using deep reinforcement learning did not only defend the cyber-physical system from zero-day attacks, but it also learned to optimize the defense policy in an accurate and timely manner.

Chapter 4

Proposed Security Solution

4.1 IDS ARCHITECTURE - Aggregation and Decentralization

The IDS architecture depicted in figure 4.1 is proposed based on the analysis of the ToN–IoT network dataset. The 45 features of the ToN-IoT dataset were reduced to 15 prime features after conducting PCA analysis and the dimensionality reduction technique of selecting the features that contribute the most variance in the dataset. Features with minuscule variance contributions were eliminated. The typical IDS is split into five decentralized IDS components. Each IDS component is trained with a unique combination of three prominent features, and every combination is different for each IDS component. For each instance of traffic that goes through the IDS architecture, it is transmitted in parallel to all IDS components to be classified as either normal or malicious traffic, based on three feature combinations that each IDS component was trained with. The classification outcome is then aggregated and a common classification result is chosen by virtue of a simple majority rule, as demonstrated in figure 4.2. Features are subsequently reshuffled during the trigger of the next shuffle iteration, which is dependent on the action taken by the agent (the defending network), as seen in figure 4.4 . It is important to state that when looking at the IDS architecture holistically as a blackbox, it is viewed as one IDS and not many IDSs. However, when it comes to its mode of operation, those IDS components come into play. The majority-rule prediction expression is given by:

$$\Sigma_{x=1}^{i}\frac{Y_{pred_x}}{n} \geq 50\% \tag{4.1}$$

where i is the total number of IDS components.

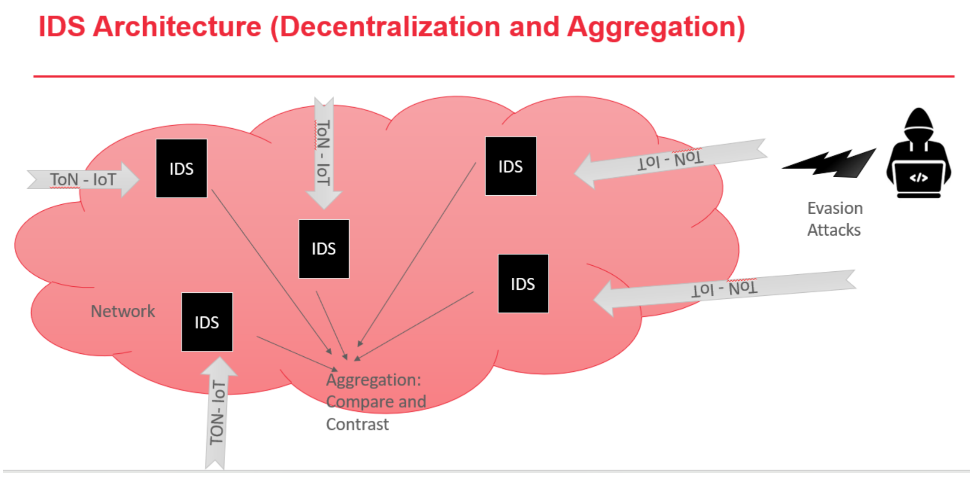

IDS Architecture (Decentralization and Aggregation)
ToN - IoT
IDS
ToN - IoT
IDS
IDS
ToN - IoT
Evasion Attacks
IDS
ToN - IoT
Network
IDS
TON- IoT
Aggregation: Compare and Contrast

FIGURE 4.1: Diagram showing IDS architecture.

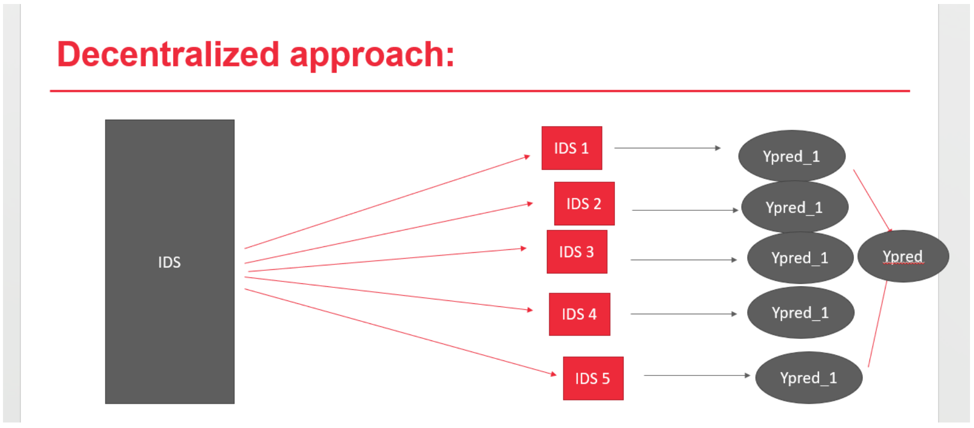


FIGURE 4.2: Diagram showing the decentralization-aggregation principle.

4.2 Threat Model

As is the case with most ML-based systems, anomaly-based IDSs are prone to adversarial attacks. The kind of adversarial attacks that the attacker is able to launch depends on the information that she is privy to about the system. The model proposed in this thesis assumes that: 1) the attacker has knowledge of the entire feature space that the IDS architecture uses to train and test traffic (data sets), 2) the attacker has knowledge of the IDS architecture including its decentralization-aggregation approach, and finally 3) she is cognisant of the splitting number of prominent features among the IDS components (which is three for this IDS architecture) figure 4.3. The only information that the attacker is oblivious to is the feature shuffling mechanism the IDS uses for the dissemination of features among IDS components. Specifically, the attacker is unaware of the exact feature combination per IDS component at any given time. Consequently if this is figured out by the attacker, he is able to inject

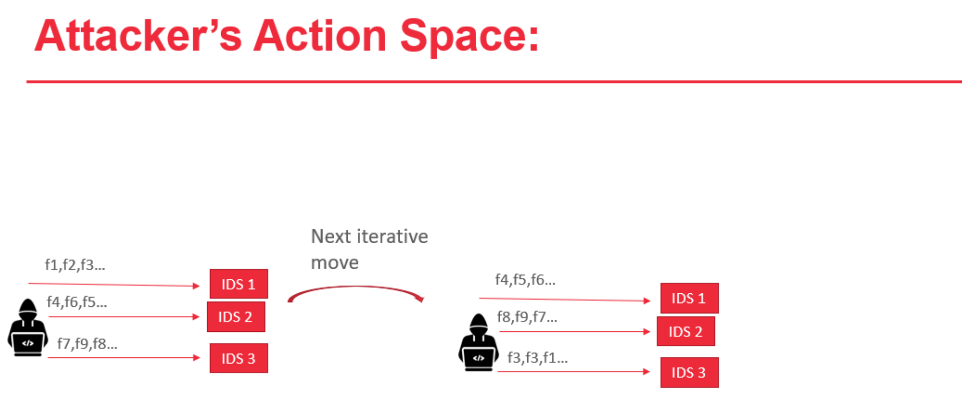


FIGURE 4.3: Diagram showing the attacker's action space.

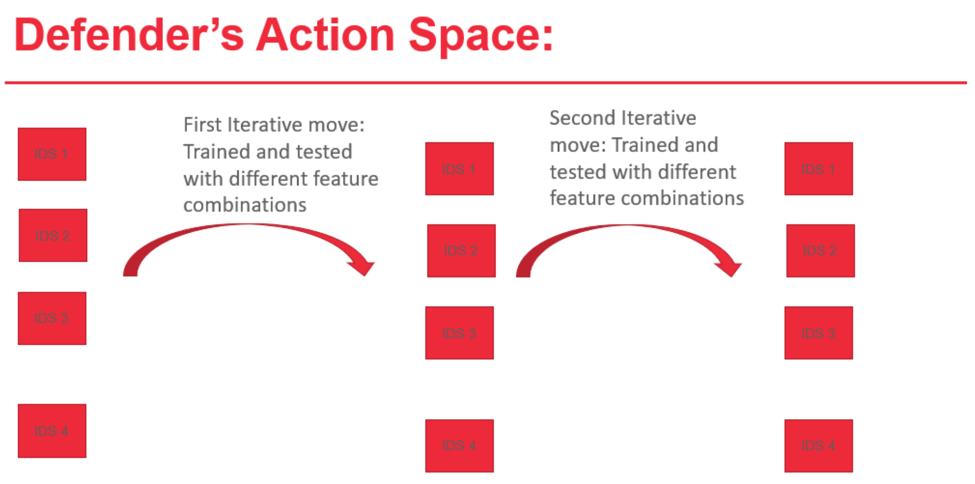

Defender's Action Space:
First Iterative move: Trained and tested with different feature combinations
Second Iterative move: Trained and tested with different feature combinations
IDS 1
IDS 2
IDS 3
IDS 4
IDS 1
IDS 2
IDS 3
IDS 4
IDS 1
IDS 2
IDS 3
IDS 4

FIGURE 4.4: Diagram showing the defender's action space.

adversarial examples into that IDS component, and the IDS component is deemed to be compromised.

Based on the information about the system which are available to the attacker, the threat model adopted for this research is the graybox adversarial attack model. It is important to note that if the attacker had knowledge of the feature shuffling mechanism, his information about the entire system would be complete and hence this would have been a whitebox attack model. Therefore, on the spectrum of graybox attack scenarios, this is the worst-case scenario from the defender's perspective. This is particularly important because in assuming that the attacker has enough knowledge to compromise a given network, we are able to address most of the loopholes within a network for worst-case eventualities which rarely happen, but are very possible.

Typically, with unrestricted access to the feature space of the network traffic of the IDS architecture, the two categories of adversarial injection attacks that could be launched are: Data poisoning and evasion attacks. The former is implemented during the training phase of the IDS (hence the name data poisoning), while the latter is usually executed in the testing phase of the IDS (real-time traffic transmission). There has been significant progress in protecting systems against data poisoning attacks and so we will only focus on evasion attacks due to their potential stealth. Also, this reduces the complexity of our model as we do not have to examine attacks during the training phase. Nonetheless, the model can always be scaled up to accommodate all types of attack scenarios in the future if it is formulated properly and systematically.

An evasion attack will typically involve manipulating a given instance of traffic so that it is misclassified by the IDS. Ideally, because the attacker has knowledge of what features the IDS uses for classification, this attack should be easy to execute over time. The shuffling of unique combination of features among IDS components however makes it difficult for the attacker to know what combination of features are used for a given IDS component at a given time t, which is in essence the objective for having this MTD-inspired architecture - to make information gathered over time by the attacker unworthy of being used.

The way the attacker is able to successfully launch an evasion attack is to constantly probe the network until she figures what combination of features are being used by the IDS components. The defending system's goal is to keep the feature shuffling process going on for as long as possible, bearing in mind the effect this has on the quality of service, system resources, and general performance of the IDS - hence the concept of a **game of features**.

Arguably the most important aspect of coming up with a working generic MTD model is the formulation of a theoretical framework to serve as a foundation for future works. This will provide further insight into the workability of the proposed solution, and ways to improve on it. To allow both the defender and the omnipotent attacker (representing innumerable attackers) to play the game of features provides insights on how long the game can go on before the entire IDS system is eventually compromised. This is the goal of MTD in any case: to stretch the time t that an attacker requires to compromise a network as practically as possible (that is, bearing in mind the cost involved) Evans and Hamkins, 2013.

4.3 Our MTD approach based on the Stochastic Game

4.3.1 Characteristics of Proposed Model

The characteristics of the proposed model can be described as follows:

- As depicted in figure 4.5, this is a dynamic game: The 5 IDS components consist of a state. They take on binary values: "1" to indicate that the IDS has been compromised, and "0" to indicate that the IDS component has not been compromised yet.
- This a semi-perfect information game: This is because the attacker is aware of when there is a change in state within the system. As compromising just one IDS component is not sufficient to launch an evasion attack (by virtue of the majority rule principle), the attacker is able to tell that his attack was impactful when his payload goes through undetected (False negatives increase significantly because at a completely compromised state, the IDS is ineffective at detecting malicious traffic). After the termination of an episode due to the game being over, the IDS architecture is triggered to change the state by the "shuffle" action, and the attacker knows this because the false negatives would have tremendously dropped after retraining. The defender on the other hand is aware of the possible move of the attacker by virtue of the fact that there is a significant disparity in the relative detection rate (True positive rate) among the IDS components. As each player has some idea of the possible move of the other, the game is not being referred to as an imperfect game. It is not a perfect information game as well because neither knows for certain the exact move of the other (only by inference) - Hence this is a semi-perfect information game Roy et al., 2010.
- This is a semi-complete information game: In complete information games the players are fully aware of the strategies and pay-offs of each other. In this game neither player has full knowledge of the other's strategies and pay-offs. The defender is able to deduce (have a fair idea) of what the attacker's expected pay-offs are based on the number of compromised nodes that resulted in the triggering of an adaptation. The attacker on the other hand is able to have a fair idea of the strategies employed by the defender based on say the number of times a particular state recurs.
- This is a stochastic game, Alpcan and Basar, 2006: One of the fundamental properties of this dynamic game model is its formulation as a stochastic process using Markov

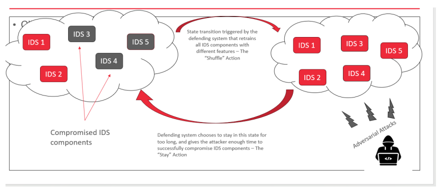

IDS 1
IDS 3
IDS 5
IDS 4
IDS 2
Compromised IDS components
State transition triggered by the defending system that retrains all IDS components with different features – The "Shuffle" Action
IDS 1
IDS 3
IDS 5
IDS 2
IDS 4
Defending system chooses to stay in this state for too long, and gives the attacker enough time to successfully compromise IDS components – The "Stay" Action
Adversarial Attacks

FIGURE 4.5: Diagram showing dynamic game scenario.

chains. Each state represents the condition of the IDS component as being either compromised or not, and the transition from one state to another, or choosing to remain in the same state can be influenced by a number of factors: one or more IDS components within a conglomerate being supposedly compromised (By inferring from the relative detection rate, α_r), the defender just trying to maximize their rewards, etc. Naturally, the number of valid actions heavily depend on the operational goals of the game, for example, allowing only feature combinations that result in a certain accuracy percentage of the overall IDS. This stochastic game is formulated so that it is possible to move from one valid state to any other valid state - Hence it satisfies the irreducible property. It is also formulated so that it is possible to return to a state after transitioning from it - Hence it satisfies the recurrent property of Markov chains. Based on satisfying these two properties, there ought to exist at least one stationary probability π when multiplied by the transition matrix P of the game that results in π.

4.3.2 Moving Parameters

Diversification:

The feature space diversification of the theoretic model highlights the different possible feature combinations of the IDS components. For i components of a given IDS architecture, each component is represented by a bit. Hence i components would be denoted by i bits. Assuming 5 IDS components (which is the exact amount used in the set up of the test bed), the IDS architecture is represented by $[00000]$ bits. The flip of a bit from 0 to 1, denotes the compromise of an IDS component. The termination of an episode is triggered when more than half the IDS components ($i/2$) have 1 as their bit values. It is important to re-emphasize that the objective is to create as much uncertainty in the system as possible, so as to make it difficult for an arbitrary attacker to successfully launch an attack over time.

Let $X = i/2$:

The diversification of the architecture itself would be the sum of combinations of all possible bits that can take the bit value 0 before termination at $X = i/2$. This is bounded by: ${}^{i}C_{i} +^{i} C_{i-1} +^{i} C_{i-2} +^{i} C_{i-3} +^{i} C_{i-4} + ... +^{i} C_{i-X}$,

where:

$$ {}^{i}C_{X} = \frac{n!}{X!(i-X)!} \tag{4.2} $$

the diversification of feature combination (the moving parameter itself), is given by: ${}^{n}C_{f}$ The total diversification of the system is thus given by the product of the two expressions:

$$ ({}^{i}C_{i} +^{i} C_{i-1} +^{i} C_{i-2} +^{i} C_{i-3} + ... +^{i} C_{i-X}) \times {}^{n}C_{f} $$

Entropy:

Entropy is a measure of the randomness or uncertainty within the system. The correlation between entropy and uncertainty implies that the greater the entropy within the system, the higher the uncertainty and difficulty for the attacker to successfully gather reconnaissance and launch an attack Zhuang, DeLoach, and Ou, 2014. The entropy $H(X)$ is given by:

$$ H(x) = -\sum_{x \in X} p(x) log(p(x)) \tag{4.3} $$

Based on the diversification expressions outlined, the maximum probability the theory stipulates for which the IDS architecture will not be compromised is given the by:

$$ ({}^{i}C_{i}(\frac{1}{2})^{i}(\frac{1}{2})^{0} +^{i} C_{i-1}(\frac{1}{2})^{i-1}(\frac{1}{2})^{1} +^{i} C_{i-2}(\frac{1}{2})^{i-2}(\frac{1}{2})^{2} + \\ {}^{i}C_{i-3}(\frac{1}{2})^{i-3}(\frac{1}{2})^{3} + ... +^{i} C_{i-X}(\frac{1}{2})^{i-x}(\frac{1}{2})^{x}) \cdot \left({}^{n}C_{f}(\frac{1}{n})^{3}\right) $$

The $\frac{1}{2}$ probability of the IDS components represents a uniform probability of a bit either being 0 or 1. The $\frac{1}{n}$ probability of the feature set represents a uniform probability of training a given IDS component from an n feature space.

The entropy then is given as:

$$ H(x) = -\sum_{j=0}^{x} \left({}^{i}C_{i-j} \left(\frac{1}{2}\right)^{j} \left(\frac{1}{2}\right)^{i-j} \cdot^{n} C_{f}(\frac{1}{n})^{f}\right) \cdot \log_{2} \left({}^{i}C_{i-j} \left(\frac{1}{2}\right)^{j} \left(\frac{1}{2}\right)^{i-j} \cdot^{n} C_{f}(\frac{1}{n})^{f}\right) \tag{4.4} $$

Based on the aforementioned expressions, we are able to ascertain the diversification and, subsequently, the entropy of our model. Our testbed is comprised of 5 IDS components($i = 5$), the maximum number of zero bits permissible X is $i/2 = 5/2 = 2.5 \approx 3$. The number of feature combinations f is 3, and n is the total number of prominent features to describe the data set and is equal to 10. Hence the diversity of our system is expressed by:

$$({}^5C_5 +^5 C_4 +^5 C_3) \times {}^{10}C_3 = (1 + 5 + 10) \times 120 = 16 \times 120 = 1920$$

To calculate the entropy, we define the probabilities as follows:

$$P_1 = \left({}^5C_5 \left(\frac{1}{2}\right)^5 \left(\frac{1}{2}\right)^0 \times^{10} C_3 \left(\frac{1}{10}\right)^3\right)$$
$$P_2 = \left({}^5C_4 \left(\frac{1}{2}\right)^4 \left(\frac{1}{2}\right)^1 \times^{10} C_3 \left(\frac{1}{10}\right)^3\right)$$
$$P_3 = \left({}^5C_3 \left(\frac{1}{2}\right)^3 \left(\frac{1}{2}\right)^2 \times^{10} C_3 \left(\frac{1}{10}\right)^3\right)$$

Then, the entropy $H(X)$ is given by:

$$\begin{aligned}H(X) &= -P_1 \log_2(P_1) - P_2 \log_2(P_2) - P_3 \log_2(P_3)\\ &= -\left(\frac{1}{32,000}\right)\log_2\left(\frac{1}{32,000}\right) - \left(\frac{1}{1,600}\right)\log_2\left(\frac{1}{1,600}\right) - \left(\frac{1}{2,500}\right)\log_2\left(\frac{1}{2,500}\right)\\ &\approx 0.1887 + 0.3752 + 0.4644\\ &\approx 1.0283\, bits\end{aligned}$$

The value above measures the entropy of our system, which provides theoretical insight on the level of uncertainty of our system. Any other system based on our model can measure the level of uncertainty of their system in comparison to another system employing the same model.

Based on these theoretic expressions, three things can be deduced:

- The measure of uncertainty can be increased by increasing the features space n from which IDS components can be trained.
- The measure of uncertainty can be increased by reducing the number of feature combinations f that each IDS component is trained with, given a feature-space pool n. This

obviously would have an impact on the accuracy of each IDS component because the less features one trains an IDS with, the less information it has to properly classify a given data instance.

- The measure of uncertainty can be increased by increasing the number of IDS components i in a given IDS architecture.

To further prove this assertion, assuming there exists another experimental testbed that employs our model but having the following parametric differences: $i = 7$, $n = 20$, $f = 3$. The termination point $i/2 = 7/2 \approx 4$. The diversity is given by:

$$({}^7C_7 +^7 C_6 +^7 C_5 +^7 C_4) \times {}^{20}C_3 = 64 \times 1140 = 7290$$

Calculating the total probabilities of each term:

$$\begin{aligned}
\text{Probability}_1 &=^7 C_7 \left(\frac{1}{2}\right)^7 \left(\frac{1}{2}\right)^0 \times^{20} C_3 \left(\frac{1}{20}\right)^3 \\
\text{Probability}_2 &=^7 C_6 \left(\frac{1}{2}\right)^6 \left(\frac{1}{2}\right)^1 \times^{20} C_3 \left(\frac{1}{20}\right)^3 \\
\text{Probability}_3 &=^7 C_5 \left(\frac{1}{2}\right)^5 \left(\frac{1}{2}\right)^2 \times^{20} C_3 \left(\frac{1}{20}\right)^3 \\
\text{Probability}_4 &=^7 C_4 \left(\frac{1}{2}\right)^4 \left(\frac{1}{2}\right)^3 \times^{20} C_3 \left(\frac{1}{20}\right)^3
\end{aligned}$$

The probabilities are given by:

$$\begin{aligned}
\text{Probability}_1 &= \frac{1}{2^7} \times \frac{20!}{3! \times (20-3)!} \times \left(\frac{1}{20}\right)^3 \\
\text{Probability}_2 &= \frac{1}{2^7} \times \frac{7!}{6! \times 1!} \times \frac{20!}{3! \times (20-3)!} \times \left(\frac{1}{20}\right)^3 \\
\text{Probability}_3 &= \frac{1}{2^7} \times \frac{7!}{5! \times 2!} \times \frac{20!}{3! \times (20-3)!} \times \left(\frac{1}{20}\right)^3 \\
\text{Probability}_4 &= \frac{1}{2^7} \times \frac{7!}{4! \times 3!} \times \frac{20!}{3! \times (20-3)!} \times \left(\frac{1}{20}\right)^3
\end{aligned}$$

Hence the entropy $H(X)$ is given by:

$$\begin{aligned} H(X) = &-(\text{Probability}_1 \times \log_2(\text{Probability}_1)) \\ &- (\text{Probability}_2 \times \log_2(\text{Probability}_2)) \\ &- (\text{Probability}_3 \times \log_2(\text{Probability}_3)) \\ &- (\text{Probability}_4 \times \log_2(\text{Probability}_4)) \end{aligned}$$

Finally, we get the approximate value of the entropy:

$$H(X) \approx 2.83169\, bits$$

It can be seen from these two performance metrics that increasing the parametric values increases the uncertainty in the system and could make one system better than the other as far as evading the attacker is concerned. It should however be stressed that it is preferred to use entropy as the ultimate performance measure of uncertainty because it provides a tighter and systematic way of looking at which parameters actually matter when it comes to increasing the uncertainty within the defending system Navas et al., 2020. For example it can be clearly seen that the number of IDS components i is the most important metric in any test bed that employs this model. Consequently increasing the number of IDS components would greatly increase the uncertainty and hence make it even more difficult for the attacker.

Having said that, it is important to also state that there is an inverse relation between parameters that increase the uncertainty in the system, and the computational and memory resources required to implement them. It is hereby portended that, for example, increasing the number of IDS components within the network i would require even more computational resources within the system to implement.

4.3.3 Game-theoretic Model of MTD Solution

For an IoT network with an MTD system defined by $\langle \vec{S}, \vec{O}, \vec{P} \rangle$, the operational Objectives can be expressed in terms of the accuracy of IDS, reduced latency, MTD overhead, and other factors. The security objectives can be expressed in terms of the number of recurring states in the model, minimum number of compromised IDS components to trigger a move, and value of relative detection in a given move trigger (intricately linked to the number of compromised IDS).

The MTD model can adopt a strict policy to ensure that the accuracy of the MTD-based system always exceeds a certain threshold (for example, $a \geq 0.98$). Also, every valid state in the model is subject to the defined objectives (security and operational) and the pre-defined policies of the game. This can be expressed as: $\forall \vec{S_v} \in \vec{S}$ subject to $\vec{O}$ and $\vec{P}$.

This game is modelled as a Markov chain as follows:

- $P(X_{n+1} = S_1 | X_n = S_0, X_{n-1} = S_2 ... X_{n-2} = S_4) = P(X_{n+1} = S_1 | X_n = S_0)$
  This fundamental Markov property implies that for this model, the probability of transitioning from X_n to X_{n+1} only depends on the probability distribution of the system as at X_n.

- There exists a transition matrix $P(|S_v| \times |S_v|)$ with (rows,columns) $\rightarrow (i, j)$, such that $\Sigma_{x=1}^{|S_v|} i = 1$. This implies that the sum of all outward probabilities (row vectors) from a given state is one.

- For an arbitrary number of transitions (moves) n_t, there exists an initial probability distribution vector π_0 at the state of zero transition, such that $P(X_0) \in \pi_0, \forall P(X_0)$.

- There exists at least one stationary probability distribution π in the game such that: $\pi \times P = \pi$, as $n_t \rightarrow \infty$. $\pi \times P = \pi \rightarrow \vec{v} \times A = \lambda \times \vec{v}$, implying that the stationary probability π is an eigenvector whose eigenvalue λ is always one. More about the implications of this to be discussed later in the thesis.

- The stationary probability distribution π is related to the initial probability distribution π_0 of the system by: $\pi_0 \times P^n = \pi$, as $n_t \rightarrow \infty$.

Consider a stochastic game model with an attacker and a defender. The model includes the following components:

- **States:** Let S_v be the set of all possible states in the game.

- **Actions:**

  - Attacker: Let A_A be the set of all possible actions available to the attacker.

  - Defender: Let A_D be the set of all possible actions available to the defender.

- **Transition probabilities:** Let $P(s'|s, a_A, a_D)$ represent the transition probability from state s to state s' given the actions of the attacker (a_A) and defender (a_D). This captures the stochastic nature of the game, where the next state depends on the current state and the actions taken.
- **Payoff functions:**
  - Attacker: Let $R_A(s, a_A, a_D)$ represent the payoff received by the attacker when in state s and taking action a_A, given the defender's action a_D.
  - Defender: Let $R_D(s, a_A, a_D)$ represent the payoff received by the defender when in state s and taking action a_D, given the attacker's action a_A.
- **Strategies:**
  - Attacker: Let $\pi_A(s)$ be the attacker's strategy, which determines the action a_A to be taken in state s.
  - Defender: Let $\pi_D(s)$ be the defender's strategy, which determines the action a_D to be taken in state s.
- **Value functions:**
  - Attacker: Let $V_A(s)$ represent the expected cumulative payoff for the attacker starting from state s, considering the attacker's strategy π_A and the defender's strategy π_D.
  - Defender: Let $V_D(s)$ represent the expected cumulative payoff for the defender starting from state s, considering the attacker's strategy π_A and the defender's strategy π_D.

The stochastic game model equation can be expressed as follows:

$$V_A(s) = \max_{\pi_A(s)} \left[\min_{\pi_D(s)} \left\{ R_A(s, a_A, a_D) + \sum_{s'} P(s'|s, a_A, a_D) \cdot V_A(s') \right\} \right] \tag{4.5}$$

$$V_D(s) = \min_{\pi_A(s)} \left[\max_{\pi_D(s)} \left\{ R_D(s, a_A, a_D) + \sum_{s'} P(s'|s, a_A, a_D) \cdot V_D(s') \right\} \right] \tag{4.6}$$

In equations 4.5 and 4.6, we take the maximum over the attacker's strategy and the minimum over the defender's strategy to capture the adversarial nature of the game EO, Alese, and Ogundele, 2013. The value functions are recursively defined, considering the expected cumulative payoffs and the transition probabilities.

This research proposes an MTD-enabled system inspired by game theory as an optimally elegant way for thorough scientific analysis in order to come up with reproducible solutions. Any game theoretic model can be further expanded to include reinforcement learning as a potential solution. This is because this machine learning paradigm is based on the trial-and-error approach, and so an agent and its environment can be modeled as playing repetitive games of trial-and-error until an optimal solution is found. If we are able to model our IDS architecture so that a given state provides information about how many IDS components have been compromised or not, and the transition to the next state is dependent on the attacker's success rate in compromising IDS components, assuming a perpetual attacker (an attacker constantly probing), and also that the transition to the next state is dependent on the defending system's choice of action (whether to stay in a state or shuffle), then the perpetual attacker's impact can be incorporated as part of the environment that the defending system (agent) has to learn from. A reward function can be modelled to reflect the state-to-state transitions (as seen in algorithm 1, and the defending system itself would be the agent in this reinforcement learning setup.

The majority rule criteria for the IDS architecture to maintain high classification accuracy implies that if more than half of the IDS components are compromised, then the game is over for the defender, and hence the termination of an episode. Hence, there would exist different ways that a game/episode could terminate, and hence reinforcement learning is used to ascertain the most optimal policy for the defender based on the testbed, as well sub-optimal strategies that could approach the optimal strategy. The next section explains the details of the proposed MTD model. Naturally, as with any MTD system, it is important to be clear about the questions: What is being moved? When is it being moved? And how is it being moved?

4.3.4 MTD Characteristics in Our IDS Model

What to move?

This is a game of features and the defender is trying to create as much uncertainty about which features a given IDS component is using at any given point in time. Each IDS component is trained with a combination of n features. n for this testbed was chosen to be 3 based on the analysis performed on the ToN-IoT dataset (which is the standard dataset that this research uses for all experiments). It was found that for an IDS component to be trained on a combination of prominent features that have enough variance for decent classification of traffic instances, they ought to be trained with at least three features. Training of the IDS is done online when a shuffle is triggered. The training is done in the background and hence consumes the system's resources. The IDS components need to collaborate with each other in other to classify a given traffic instance. Therefore, they ought to be trained with the same instances for maximum results. The only difference is that each IDS component's classifier interprets a given traffic instance based on the feature combination assigned to it, but the traffic instances must be the same for all. This could be described as different ways of looking at the same thing, and then we aggregate all perspectives to get the full picture as to whether a traffic instance is malicious or benign - The majority rule criterion. Hence, for a given system being attacked, there is one entity being moved around: Features. The shuffle of features among IDS components makes a given feature combination a moving parameter.

When to move?

If each IDS component has a true positive rate α (one of the parameters from Table 4.1, which is from here on referred to as the detection rate) Liu, Comaniciu, and Man, 2006, and if the attacker is assumed to have unlimited resources (hence a representation of any arbitrary number of attackers) and is constantly bombarding IDS components with data instances that have different combination of features with the aim of finding out what combination of features is being used for a given IDS component, then it is expected that the detection rate of IDS components would be relatively high. If one or more IDS components within the architecture start to record detection rates that are relatively lower than other components, this implies that it may have been compromised and the attacker has been able to evade detection. When half or more IDS components have been deduced to be compromised, this triggers the conglomerate (IDS architecture containing IDS components) to shuffle features and re-train IDS components - This process is deemed as the last move before the end of an episode. It is important to note that when an IDS component is deemed compromised and flips from 0 to 1, this also changes the state of the conglomerate (architecture) and so is technically considered a move in its own right. The only difference is that this kind of transition/move caused by the environment itself (as the attacker's malicious activities have been incorporated into the environment as a key element) holds little relevance to the actual purpose of adopting an MTD model in the first place - Which is to move the attack surface of a given IDS architecture so that it is difficult over time for an attacker to successfully complete an attack.

The relevant state transitions would be the transitions triggered by the agent itself (the defending system). This is the main reason reinforcement learning was incorporated into the model: The agent's decision to trigger a shuffle or not would either result in an episodic game ending quickly or going on for as long as possible. An episode would go on as long as possible provided the agent chooses to shuffle before at least half of the IDS components within the conglomerate are compromised. However, choosing to shuffle continuously results in computational and memory costs on the IDS architecture because to shuffle and reshuffle implies training and retraining IDS components with different feature combinations iteratively for every given shuffle trigger move. Ultimately, the agent would want to learn about the environment over time, through trial and error, for a given number of episodes and total corresponding rewards yielded based on the actions that were taken, and to find the closest optimal policy or sub-optimal policy (as this is subject to the number of episodes it has experienced), so as to determine when to initiate a move.

How to move?

A huge discrepancy in detection rates α_r (see table 4.1) where for example one or more IDS components start to record very low detection rates (True positives), and consequently high true negatives, is taken to imply that the attacker has been successful in deciphering the feature combination used for the IDS component/s and hence is evading detection. The threshold for uniform detection rates among IDS components within a conglomerate ought to be determined based on the specific system and network.

The IDS architecture/conglomerate is retrained online in the background, which has significant computational and memory implications (costs), and features are reshuffled among IDS components. Each IDS component is assigned a combination of features that could be constrained by the total accuracy criterion set. For example, if it is desired that after training and testing, the accuracy obtained from aggregating the results of each IDS component should not be less than 97% , then this obviously restricts the number of combinations that are possible - These would be referred to as valid combinations. After reshuffling and training is complete, the conglomerate's current IDS components are replaced in real time with the newly trained set.

The reward function of the agent depends on the number of compromised components i_c, the number of shuffled components $i_s = i - i_c$, and the total number of IDS components within a given architecture i. It also depends on the action taken (whether "shuffle" or "stay"), and if the action is taken at a time when the number of compromised IDS components i_c is below or above the threshold $i/2$.

4.4 Reinforcement Learning Structure for Solving the MTD Timing Problem

Based on this model, the Defender has two (2) actions: Stay or Shuffle. The "Stay" action indicates a decision by the defender to remain in the same state based on pre-defined objectives of the model. For example: Reward threshold, number of compromised IDSs, etc. The "Shuffle" action indicates a decision by the defender to change state. This also means the "Shuffle" action would involve different feature combinations and how that affects metrics like the overall performance of the IDS architecture. Algorithm 3 shows what transpires within an ensuing game when the agent takes an action.

For the attacker's actions, the model is set up so that the attacker constantly attempts to launch a successful evasion attack. This would naturally consist of sub-actions of feature combinations. It is important to note that the attacker's action set can be expanded (e.g.,

TABLE 4.1: Parameters and notations used in the game-theoretic model.

Parameter definition	Notation
Number of IDS components in the IoT network	i
True positive rate of an IDS component	α
Relative detection rate of IDS components within a conglomerate	r
Stationary probability distribution of stochastic game	π
Transition Matrix	P
Number of states in system	S
Number of valid states	S_v
Initial probability distribution of stochastic game	π_0
Compromised IDS components per state	i_c
Number of shuffled IDS components per state	i_s
Number of prominent features	f_p
Number of redundant features	f_r
Total number of features in feature space	n
Accuracy policy of system	a
All actions available to attacker	A_A
All actions available to defender	A_D
A given action of the attacker	a_A
A given action of the defender	a_D
Attacker's given strategy	$\pi_A(s)$
Defender's given strategy	$\pi_D(S)$
Number of unique features per IDS component	f

"Don't Evade" or "Evade only when a certain attacker reward threshold is reached"). For the sake of simplicity of the model, the attacker's action is to constantly attempt to evade, and is incorporated into the environment the agent (defender) has to learn.

Given an arbitrary pool of prominent features to shuffle from f_p, if each IDS component was shuffled with f features, then the number of combinations possible for each IDS component from the pool is given by: ${}^{f_p}C_f$. To represent each IDS components' combination by using bits, we obtain $\log_2({}^{f_p}C_f)$ bits. Hence assuming 8 prominent features to shuffle from, taking 3 features at a time to train on IDS components, then ${}^{8}C_3 = 56$. With 56 combinations per IDS component, each IDS component can be represented with: $\log_2 56$ bits.

This relation indicates that each IDS component's feature combination is represented as an x-bit binary number, which is extremely important in the implementation phase, where it is much easier and more convenient to put an abstraction on feature combinations with binary representations. Hence, an evasion attack is deemed successful if for any arbitrary $\log_2({}^{f}C_{f_{split}})$-bit feature combination of each IDS component, the attacker's feature combination matches.

The transition matrix P is the mathematical representation of the environment in which the defender operates. Based on the initial goals stipulated for the game, P can vary across different experimental trials. To obtain P, the concept of reinforcement learning is used, where the defender blindly initiates moves and gets rewards and tries to maximize rewards based on trial and error. It is important to indicate that the evasion attacks being launched by the attacker is deemed to be part of the environmental space in the eyes of the defender. Hence, from the defender's perspective, we only have one agent performing reinforcement learning. The probability of moving from one state to another given a certain action is given as:

$$\hat{T}(s,a,s') = \frac{*times(s,a,s')}{*times(s,a)} \tag{4.7}$$

Having obtained the transition Matrix P ($\hat{T}(s,a,s')$) of the game, a reinforcement learning solution can be formulated. Given a policy function $\pi(s)$, which indicates a set of "action-paths" to follow as a function of the present state s, $V\pi(s)$ is the expected utility at a given state and $U(R,\gamma)$ is the sum of discounted rewards (where γ is the discount factor and is equal to 1 if the reward function is not affected by the circumstances of the future). Figure

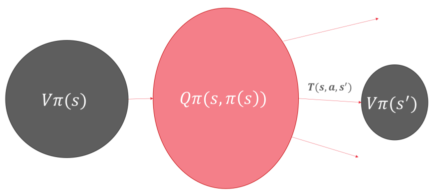


FIGURE 4.6: A diagrammatic representation of the stochastic game equation.

4.6 gives a pictorial appreciation of the aforementioned RL solution Brunton and Kutz, 2022:

$$Q(s,a) = E[R(s,a,s') + \gamma V\pi(s')] \tag{4.8}$$

$$Q(s,a) = \sum_{s'} \hat{T}(s,a,s')[R(s,a,s') + \gamma V\pi(s')] \tag{4.9}$$

$$V\pi(s) = max_a Q(s,a) \tag{4.10}$$

$$V\pi(s) = max_a \sum_{s'} \hat{T}(s,a,s')[R(s,a,s') + \gamma V\pi(s')] \tag{4.11}$$

$$\pi(s,a) = argmax_a Q(s,a) \tag{4.12}$$

Algorithm 1: Algorithm showing reward function of the agent

```
Input  : $i_c$, $i$
Output: reward
Function compute_reward():
    reward ← i - i_c;
    if agent_action == 1 and i_c ≤ i/2 then
        cost ← a * i_c;
        return (reward - cost);
    end
    else if agent_action == 1 and i_c > i/2 then
        cost ← b * i_c;
        return (reward - cost);
    end
    else if agent_action == 0 and i_c ≤ i/2 then
        cost ← c * i_c;
        return (reward - cost);
    end
    else if agent_action == 0 and i_c > i/2 then
        cost ← d * i_c;
        return (reward - cost);
    end
```

The RL problem will be solved by recurrence. $Q\pi(s, a)$ is referred to as the q value at a chance node and is purely abstract. It is only relevant to help explain the intricacies of the equation above (that can seem counter-intuitive at first glance). It is the quality of a state action-pair. Essentially, the optimal value at any state s would be the Q-value that is maximized over a set of actions. The optimal policy $\pi(s, a)$ in this case is the state-action combinational subset that maximizes the quality function (Q-value), as seen in algorithm 4.

The bell's optimality equations 4.9 and 4.11 are feasible solutions only when there is a working model for the environment, that is, generating a transition matrix based on the probability distribution of state transitions. In reality, this is often not the case. As in any real-world scenario of how humans learn, understanding how to go about a task requires learning exclusively by trial and error. This is where model-free reinforcement learning solutions come in handy.

The first model-free reinforcement learning technique to be considered is the Monte-Carlo technique:

$$\sum R = \sum_{i=1}^{n} \gamma^i r_k \tag{4.13}$$

Basically this method evaluates the cumulative rewards over various episodes (state-actions) after termination.

$$Q^{new}(s_i, a_i) = Q^{old}(s_i, a_i) + \frac{1}{n}(\sum R - Q^{old}(s_i, a_i), \forall i \in [1...n] \tag{4.14}$$

The Monte Carlo method updates the previous Q-value of a given state based on positive average differences between old Q-values and the cumulative reward of a given episode. Taken the average assumes that for any chosen optimal or sub-optimal policy, the net gain in rewards from every state involved in the policy until termination is equal, but this is practically not the case. This eliminates bias, but it is quite inefficient in how it extracts the optimal policy. In principle with enough iterations, the value at any state using the Monte Carlo approach should always converge to the optimal.

The next reinforcement learning technique to be considered is $Q - learning$. Unlike Monte-Carlo, this technique has information about which state-action in any given policy yields the most rewards. At each iterative step, there is a re-evaluation of the value of a state (or the quality of a state-action pair Q) until termination.

$$Q^{new}(s_i, a_i) = Q^{old}(s_i, a_i) + \alpha]r_i + \gamma max_a Q(s_{k+1}, a) - Q^{old}(s_i, a_i)], \forall i \in [1...n] \tag{4.15}$$

Essentially Q-learning is an off-policy reinforcement learning technique. This implies that the quality of the next state-action pair is explorative, and will capture even sub-optimal Q-values in an attempt to find the optimal policy, which is the argmax that maximizes the Q-value for the next state. Its off-policy quality makes it possible to keep track of all (or almost all) episodes that lead to termination (from optimal policies to sub-optimal policies to below optimal policies), there is a complete perspective of learning about the environment. Depending on one's preference for exploration, there is a tunable parameter ϵ to add some randomness in exploring sub-optimal policies.

State–action–reward–state–action (SARSA) is very similar to Q-learning, except that this technique is on-policy, and each next state-action pair has to be optimal or else there would be a constant degradation of the Q-value of a given state-action pair.

$$Q^{new}(s_i, a_i) = Q^{old}(s_i, a_i) + \alpha]r_i + \gamma Q(s_{k+1}, a_{k+1}) - Q^{old}(s_i, a_i)], \forall i \in [1...n] \quad (4.16)$$

Depending on one's preference and testbed setup, one could adopt any of the reinforcement learning solutions discussed above to carry out experiments on our MTD model. The reinforcement learning solution used for this project is Q-learning (with a temporal difference learning of 0, $TD(0)$), as outlined in the aforementioned Q-learning equation 4.15, and the Q-learning algorithm in algorithm 2).

Algorithm 2 shows how the MTD model was implemented using Q-learning as the wrapping framework. The Q-table is updated iteratively, and the agent's choice of action gains optimality over time.

Algorithm 2: Algorithm for reinforcement learning framework (Q learning) of MTD model

```
Input  : state space, action space, q table, alpha, gamma, epsilon, num-episodes,
         max-steps-per-episode, iteration-count
Output: old-state, new-state, agent-action, reward, accuracy
$state\_space \leftarrow 32$;
$action\_space \leftarrow 2$;
$q\_table \leftarrow$ np.zeros($state\_space$, $action\_space$);
$\alpha \leftarrow 0.5$;
$\gamma \leftarrow 0.9$;
$\epsilon \leftarrow 0.1$;
$num\_episodes \leftarrow 100$;
$max\_steps\_per\_episode \leftarrow 2500$;
$iteration\_count \leftarrow \{\}$;
$i \leftarrow 0$;
for $episode$ to $num\_episodes$ do
    $observation\_action\_pair \leftarrow []$;
    $observation\_action\_observation \leftarrow []$;
    $Game.reset(restart = True)$;
    $state \leftarrow$ binary_to_decimal(old_state) $-1$;
    for $step$ to $max\_steps\_per\_episode$ do
        if *random.uniform()* $< \epsilon$ then
            $agent\_action \leftarrow$ agent_action_space.sample();
        end
        else
            $agent\_action \leftarrow$ argmax($q\_table[state]$);
        end
        $observation, agent\_action \leftarrow$ Game.step
          (Game.agent_action_space.sample());
        $pair \leftarrow old\_state, agent\_action$;
        $observation\_action\_pair$.append($pair$);
        $triplet \leftarrow old\_state, agent\_action, new\_state, reward, acc.$;
        $observation\_action\_observation$.append($triplet$);
        $state \leftarrow$ binary_to_decimal(observation) $-1$;
        $next\_state \leftarrow$ binary_to_decimal(new_state) $-1$;
        $td\_error \leftarrow reward + \gamma \cdot$ (np.max$(q\_table[next\_state]) -$
          $q\_table[state][agent\_action])$;
        $q\_table[state][agent\_action] \leftarrow$
          $q\_table[state][agent\_action] + \alpha \cdot td\_error$;
        $state \leftarrow next\_state$;
        if *Game.done1* then
            $iteration\_count[i + 1] \leftarrow observation\_action\_observation$;
            $i \leftarrow i + 1$;
            break;
        end
    end
end
```

Algorithm 3: Step Function - Algorithm that shows the method initiated when the agent triggers an action

```
Input  : 0("Stay"), 1("Shuffle")
Output: old_state, agent_action, new_state, reward
Function step(agent_action):
    if agent_action == 1 then
        old_state ← new_state;
        reset();
    end
else if agent_action == 0 then
    old_state ← state_new; self.attack_info ← [];
    for i 0 to i do
        attack_probability ← random.uniform();
        if attack_probability $\leq$ new_state[i] == 0 then
            adversarial_attack(i);
            old_state[i] ← 1;
            i_c + 1;
        end
    end
    else
        print("Agent Action Error, please indicate either a shuffle - 1, or stay - 0");
    end
    reward ← compute_reward();
    if i_c $\geq$ i/2 then
        done ← True;
    end
    return old_state, agent_action, new_state, reward
end
```

Algorithm 4: Algorithm showing the optimal MTD process after learning

```
Input   : None
Output: None
observation_action_pair ← [];
observation_action_observation ← [];
total_reward ← 0
 iteration_count_optimal ← {};
reset(restart=True);
state ← binary_to_decimal(old_state);
for step 0 to max_episode_step do
    action ← argmax(q_table_new[state]);
    old_state, agent_action, new_state, reward ← Game.step(action);
    pair ← observation, agent_action;
    observation_action_pair.append(pair);
    triplet ← old_state, agent_action, new_state, reward;
    observation_action_observation.append(triplet);
    state ← binary_to_decimal(observation);
    next_state ← binary_to_decimal(observation_new);
    total_reward += reward;
    if Game.done1 then
        iteration_count_optimal[i+1] ← observation_action_observation;
        break;
    end
end
print("Total reward");
```

Chapter 5

Implementation using the ToN-IoT Dataset

5.1 Dataset Background

Most of the IoT datasets for intrusion detection are derivatives of benchmark network security datasets like the KDDCUP'99 and the NSL–KDD Tavallaee et al., 2009, which both have their data records synthetically generated, and hence may not be reflective of real-world scenarios. These datasets also limit their scope to a handful of attack scenarios. This is why this thesis will adopt the ToN-IoT dataset. This is a relatively recent dataset that was generated between 2020 and 2021 by Booij et al., 2021 (A group of Australian researchers from the IoT Lab of the UNSW Canberra Cyber), and is based on a realistic IoT test bed.

The ToN-IoT dataset is the data specimen for all experimental analyses conducted under this thesis. The dataset repository contains 23 separated csv files, each containing approximately one million (1,000,000) instances. The concatenation of all these into one single csv file for potability and convenience resulted in one composite dataset of 23 million instances (23,000,000) and forty-five (45) attributes. For simplicity, this was sampled to a much smaller dataset of just over a million instances, that still contained a uniform distribution of the various types of data instances and their classification types.

The authors in Booij et al., 2021 highlighted the intricacies of generating the network-layer ToN-IoT dataset. As stated before, this dataset consists of forty-five (45) attributes, which includes a classification attribute that has ten (10) different values: normal, scanning, DoS, injection, DDoS, password, xss, backdoor, ransomware, and MITM (man in the middle). These are essentially categorized as normal and malicious traffic using the label attribute. The authors in Booij et al., 2021 examine the ToN-IoT dataset by conducting the Pearson correlation coefficient (PCC) technique on the attributes to ascertain which attributes have the strongest correlation, and then analyzing the information gain on each attribute, subject to the classification attribute. They further conducted cross-validation of the ToN-IoT dataset with another IoT dataset developed from a realistic testbed called the Aposemat IoT-23 on three different supervised classification models: Gradient Boosting Machine, Random Forest, and Multilayer Perceptron (which is a class of feedforward Neural Network).

5.2 Experimental Analysis on the ToN-IoT Dataset

5.2.1 A Theoretical Framework for Investigating the ToN-IoT Dataset

The ToN-IoT dataset was investigated to ascertain if it can be used to train an anomaly-based intrusion detection system of a network which will have its parameters like IP addresses and port numbers continuously moving. To be able to do this, it was required that attributes containing IP addresses and port numbers were tinkered with so as to find out which of these attributes, if shuffled, will yield as close to the accuracy obtained from using the default dataset as possible. This is to help draw the conclusion that if the dataset was used to train an intrusion detection system whose network had dynamically changing IP configurations, the intrusion detection system will still be able to distinguish normal traffic from malicious traffic.

However, the ToN-IoT dataset has forty-five (45) attributes, which will make it computationally tasking to run multiple iterative trials of various mutations of the original dataset to ascertain if shuffling of the IP and/or port attribute values will render the dataset usable or unusable to train any anomaly-based IDS that is on a moving target network, hence the need for feature reduction.

Feature reduction using PCA: For the ToN – IoT dataset with n number of instances and m number of attributes, this results in a $m \times n$ data matrix X , with vectors $\vec{x_i} \in R^m$. We compute the covariance S of matrix X , where $S \in R^{(m \times m)}$ as follows

$$S = \frac{1}{n} \sum_{i=1}^{n} (\vec{x_i} - \bar{x}) (\vec{x_i} - \bar{x})^T \tag{5.1}$$

where $\bar{x} \in R^m$ is the mean of each row instance in X defined by:

$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} \vec{x_i}$

To extract principal components, the Singular vector decomposition of S is employed:

$$S = U\gamma V^T \tag{5.2}$$

where $U \in R^{n \times n}$, $\gamma \in R^{n \times m}$, $V \in R^{m \times m}$. The vector $V = [u_1, u_2, \ldots, u_m]$, where $u_i \in R^m$ denotes a principal component's direction.

We need to project the ToN-IoT dataset, denoted by the matrix X into a new matrix Y (where $Y \in R^{k \times m}$); this is achieved by multiplying by a matrix P^T as follows:

$Y = P^T X$ Where $P = [u_1, u_2, \ldots, u_k]$, $k \leq m$

Let the ToN-IoT dataset be denoted by matrix X, with n number of instances and k number of attributes, where $k < m$ after applying PCA. X is split according to $\frac{n}{\theta}$, where θ is the number of splits of the total instances n:

$X = \sum_{i=1}^{\theta} X_i$

Taking an arbitrary feature $\beta \in m$, we have:

For $X_i \in X$, $\beta \in \vec{x_i}$ of X_i is encoded differently to denote a shuffle: $X_i = X_i'$. Therefore, we have:

$$X = X', \quad X' = \sum_{i=1}^{\theta} X_i'$$

5.2.2 Algorithm Design and Implementation

As mentioned earlier, to reduce the computational burden, the ToN-IoT dataset was sampled to a dataset of approximately a million instances that are of a uniform distribution of the original composite dataset. Next, we perform feature reduction using the principal component analysis (PCA) technique and check for features that contributed the most variance to the dataset, as seen in figure 5.1. After the features were selected, the computational burden of tinkering with the positions of attribute values (moving/shuffling IP and/or port numbers) was done with much better ease. This was implemented using algorithm 5.

Assuming satisfactory accuracy could not be obtained, it would be concluded that the ToN-IoT dataset cannot be used to train an MTD-based intrusion detection system (IDS) on a network. If satisfactory accuracy is obtained, the attributes which had their values shuffled would be highlighted as potential parameters to move on a network whose IDS was trained on the ToN-IoT dataset. The conclusion then to be drawn if this was the case is that assuming an IoT network could be set up that has an anomaly-based IDS which was trained on the ToN-IoT dataset, it should be possible to dynamically re-configure network parameters like IP addresses to make it hard for the attacker to understand what is going on in the network, while at the same time not affecting too much the accuracy of the IDS in being able to distinguish normal traffic from malicious traffic.

The Machine Learning algorithm on which the IDS is to be set up is the Random Forest (RF) Classifier. Hence, the RF supervised classification model (with 10 decision trees) was used for the experiments carried out with the ToN-IoT dataset. Firstly, the original dataset is trained and tested, and the accuracy is measured. Then, the selected feature as the moving parameter is shuffled, and the model trained with the original dataset is tested on this newly modified dataset. The accuracy is also measured. Next, the model is trained and tested on the dataset, but this time with the selected attribute completely removed. The accuracy is measured and compared with the accuracy of the original dataset. Based on the results, the analysis is conducted, and the conclusions drawn.

Algorithm 5: algorithm showing MTD-feasibility experiments on the ToN-IoT dataset

```
Input  : Dataset with features
Output: Accuracy results
Function Preprocessing:
    // Apply PCA for feature reduction
    Apply PCA (Principal Component Analysis) to the dataset;
    // Select prominent features using variance
       techniques
    Compute the variance of each feature;
    Select features with high variance as prominent features;
    // Return the reduced feature set
    return Reduced feature set;
Function TrainTestWithX:
    // Split dataset into training and testing sets
    Split the dataset into 80% training set and 20% testing set;
    // Train Random Forest Model with 80% training set
    Train a Random Forest Model using 80% of the dataset;
    // Test the model with 20% testing set and record
       accuracy
    Test the model using the remaining 20% of the dataset and record the accuracy;
    // Return the accuracy
    return Accuracy;
Function TrainTestWithX':
    // Split dataset into training and testing sets
    Split the dataset into 80% training set and 20% testing set;
    // Train Random Forest Model with 80% training set
    Train a Random Forest Model using 80% of the dataset;
    // Test the model with 20% of X' and record accuracy
    Test the model using 20% of X' and record the accuracy;
    // Return the accuracy
    return Accuracy;
```

Algorithm 5: algorithm showing MTD-feasibility experiments on the ToN-IoT dataset

```
AnalyzeOutcomes // Compare accuracy of X and X′
Compare the accuracy of X and X’;
// Draw conclusion on attribute shuffling
Analyze the outcomes and draw conclusions on whether attribute shuffling
  maintains the accuracy integrity of the intrusion detection system;
RemoveBeta // Remove $\beta$ in Xi of X
// Split X′ into training and testing sets
Split X’ into 80% training set and 20% testing set;
// Train Random Forest Model with 80% training set
Train a Random Forest Model using 80% of X’;
// Test the model with 20% testing set and record
   accuracy
Test the model using the remaining 20% of X’ and record the accuracy;
// Return the accuracy
return Accuracy;
// Main code
Preprocessing;
AccuracyX ← TrainTestWithX;
AccuracyX’ ← TrainTestWithX’;
AnalyzeOutcomes;
AccuracyX’RemovedBeta ← RemoveBeta;
```

Metric	Number of Instances
True Negatives	118708
False Positives	0
False Negatives	3
True Positives	180179

TABLE 5.1: Table shows the confusion matrix based on the accuracy of 0.998.

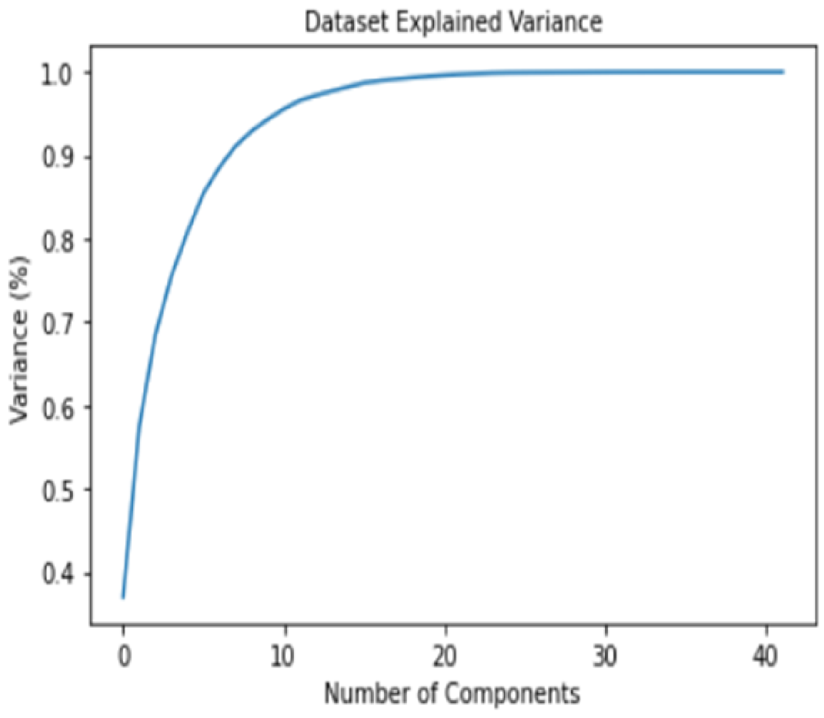


FIGURE 5.1: The diagram shows a plot of the variance that each attribute contributes to the dataset after PCA analysis. It can seen that just about 15 attributes are sufficient to describe the entire dataset.

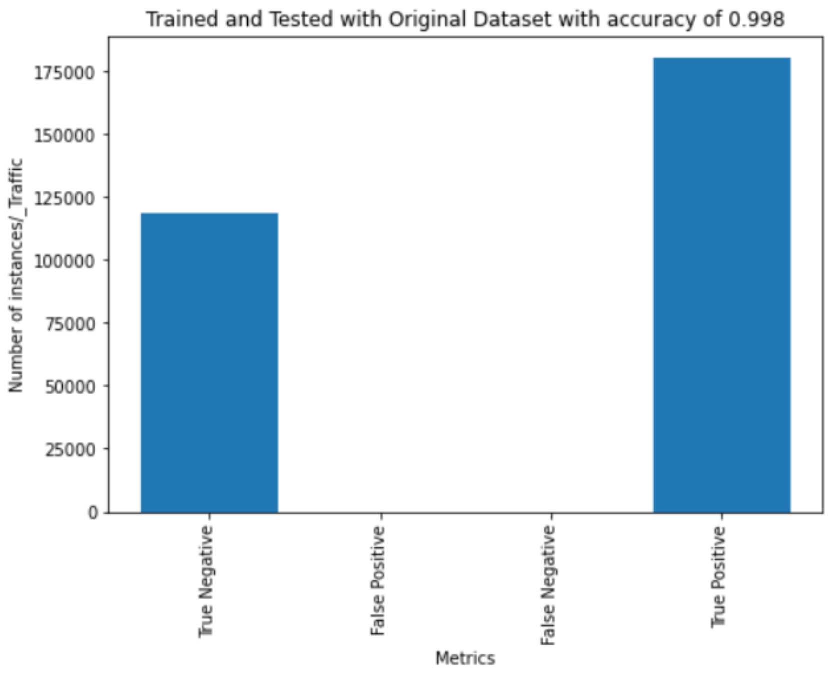


FIGURE 5.2: Results of trained and tested RF model with the original dataset (accuracy of 0.998).

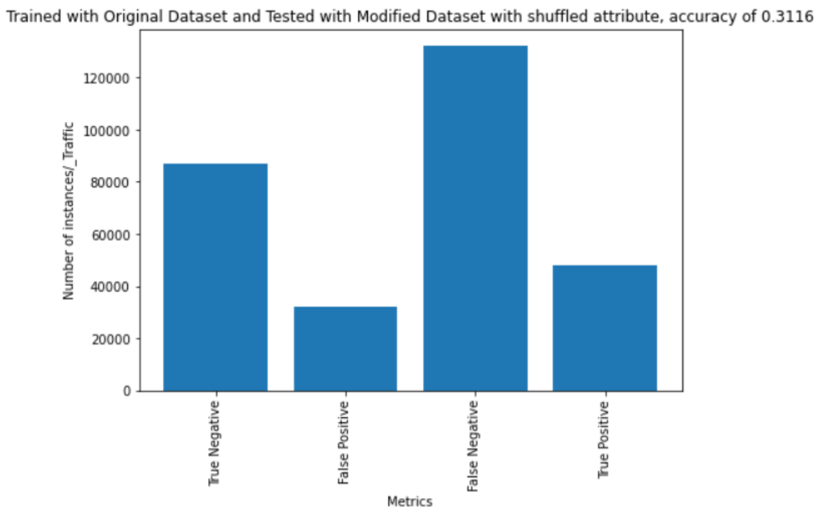


FIGURE 5.3: Results of the trained RF model with the original dataset, tested with the modified dataset with shuffled attribute (accuracy of 0.3116).

Metric	Number of Instances
True Negatives	86811
False Positives	31897
False Negatives	132141
True Positives	48041

TABLE 5.2: Table shows the confusion matrix based on the accuracy of 0.3116.

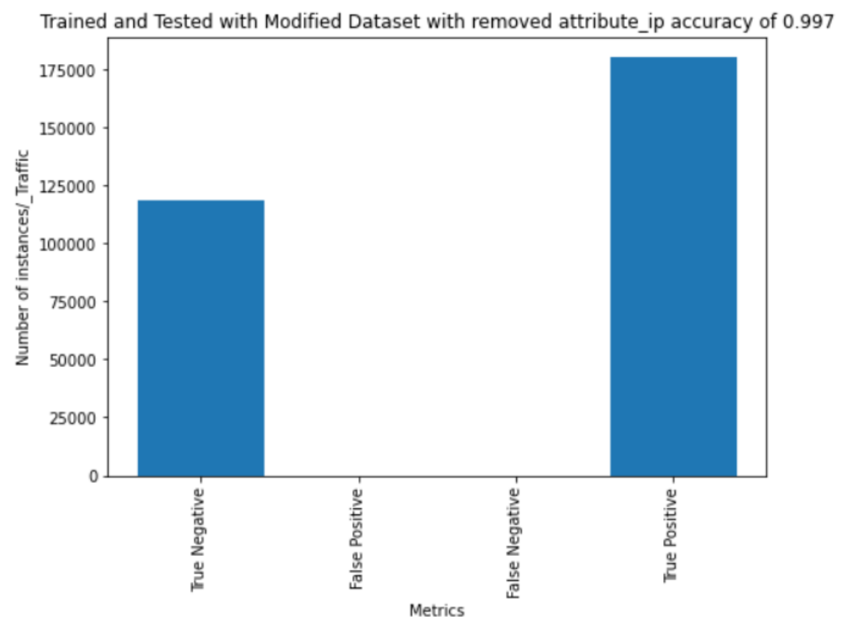


FIGURE 5.4: Results of the trained and tested RF model with the modified dataset with removed attribute (IP). Accuracy of 0.997.

Metric	Number of Instances
True Negatives	118719
False Positives	1
False Negatives	4
True Positives	180166

TABLE 5.3: Table shows the confusion matrix based on the accuracy of 0.997.

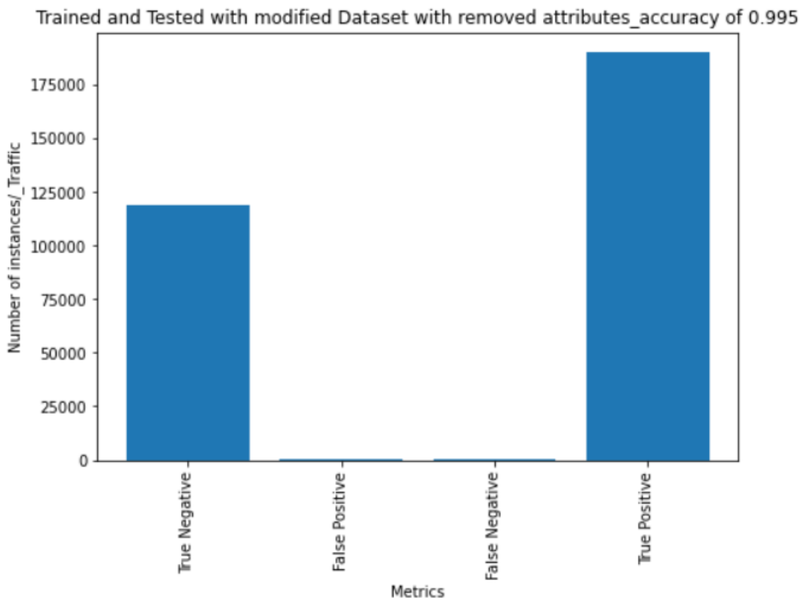


FIGURE 5.5: Results of the trained and tested RF model with the modified dataset with two removed attributes (both source IP and source ports). Accuracy of 0.995.

Metric	Number of Instances
True Negatives	118886
False Positives	86
False Negatives	106
True Positives	179812

TABLE 5.4: Table shows the confusion matrix based on the accuracy of 0.995.

5.2.3 Comments Drawn from the Experimental Analysis of the ToN-IoT Dataset:

Following the invetsigation of the dataset for the feasibility of deploying our proposed MTD approach, the below comments are drawn:

- The prediction accuracy of the IDS when trained on ToN-IoT dataset is very good in the scenario where the original dataset was used without shuffling attributes (non-MTD scenario), as seen in figure 5.2 and table 5.1.
- It is not prudent to train the IDS with a parameter/attribute whose values will be changing (by moving or shuffling), as seen in figure 5.3 and table 5.2.
- The IDS can be trained without the changing/moving attribute and still achieve a high degree of accuracy due to the fact that the attribute has a very low variance after PCA analysis of the dataset (as seen in figure 5.4 and table 5.3). This is the inspiration for the decentralization and aggregation approach of the IDS based on feature splitting that characterizes the proposed MTD model of the project.
- Both attributes (Src-Ip and Src-Port) can be removed with a high degree of accuracy achieved. Again, due to the fact that they have a very low variance after PCA analysis of the dataset (as can be seen in figure 5.5 and table 5.4).

Chapter 6

Experiments and Results

6.1 Testbed Setup and Simulations

The anomaly-based IDS relies primarily on a machine learning classifier that operates using a machine learning model. For this project, we set a random forest classifier consisting of ten decision trees as our primary ML model for the IDS. The random forest is a meta estimator that understudies and fits a given dataset into two or more decision trees, and uses the average classification of each decision tree for a more accurate and robust classification.

The MTD-based testbed consists of 5 different random forest classifiers (known as the IDS components in the proposed architecture), each containing 10 decision-tree estimators. After feature selection was conducted on the ToN-IoT dataset using PCA, the chosen features were recorded in a mutable list (called the pool of features). During the training phase of the IDS components, each was randomly trained on three unique combinations of features from the feature pool.

To compromise a given IDS component, we use the CleverHans library for our adversarial learning attack. The CleverHans library is an open source python library for conducting adversarial attacks on machine learning models. Adversarial examples were generated by feeding the test traffic through the cleverhans.torch.attack.noise class, which generated the adversarial examples for the ToN-IoT test data samples by injecting adversarial noise that led to misclassification of data instances.

Python is the main programming language used in the experimental phase. From the python repository, the Scikit-Learn library was used in the implementation of the random forest classifier model. The Pandas library was used for reading the ToN-IoT dataset as a data frame into the working environment, and then for the various manipulations of the data frame (dataset) that have already been discussed. The Numpy library was used for handling data fragments as arrays, tinkering with them, and re-writing them back on to the data frame (dataset) for further analysis. Jupyter Notebook is the working environment in which all python code was ran and set up.

Th machine used has the following specifications:

- Processor: Intel64 Family 6 Model 165 Stepping 2; GenuineIntel
- Processor Speed: $\sim$2496 MHz
- Total Physical Memory: 7,968 MB
- Available Physical Memory: 1,708 MB
- Virtual Memory (Max Size): 16,160 MB
- Virtual Memory (Available): 5,309 MB
- Virtual Memory (In Use): 10,851 MB

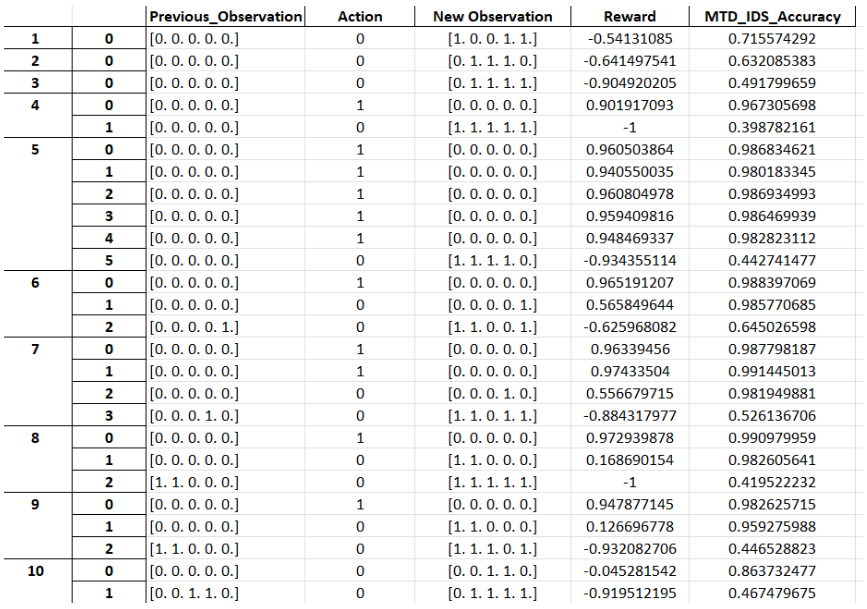
The OpenAI gym library was used to provide the baseline environment for implementation of the reinforcement learning algorithm that proactively tried to optimize the agent's (MTD-based IDS) actions.

The experimentation steps can be summarized as follows:

- Set up and train two separate IDSs: A conventional IDS and the MTD-based IDS, each using the random forest ML model as the classifier, using the ToN-IoT dataset as the traffic specimen.
- Measure and record the accuracy test traffic going through the conventional IDS. Using the cleverhans adversarial framework, inject adversarial noise into the test traffic and measure the accuracy drop, then do this for increasing odds of success of the attack launch (i.e., $0.5, 0.6, 0.8, 1$).
- Measure and record CPU and memory usage on the conventional IDS during the adversarial attack.
- Set up the reinforcement learning environmental space that inhabits the MTD nature of the MTD-based IDS. Incorporate the incessant adversarial attacks as part of the the IDS agent's environment.

- For each adversarial attack attempt on the MTD-based IDS, the RL algorithm learns the corresponding reward obtained progressively over time and influences in real time what action the agent takes over the course of a number of episodic runs. The odds of the attacker are increased accordingly from $50\% \ to \ 100\%$. The accuracy at each of these odds is recorded, as well as the CPU and memory usage amidst adversarial attacks and MTD deployment.

- The results of implementing a conventional IDS and an MTD-based IDS are compared and contrasted, and conclusions are drawn.

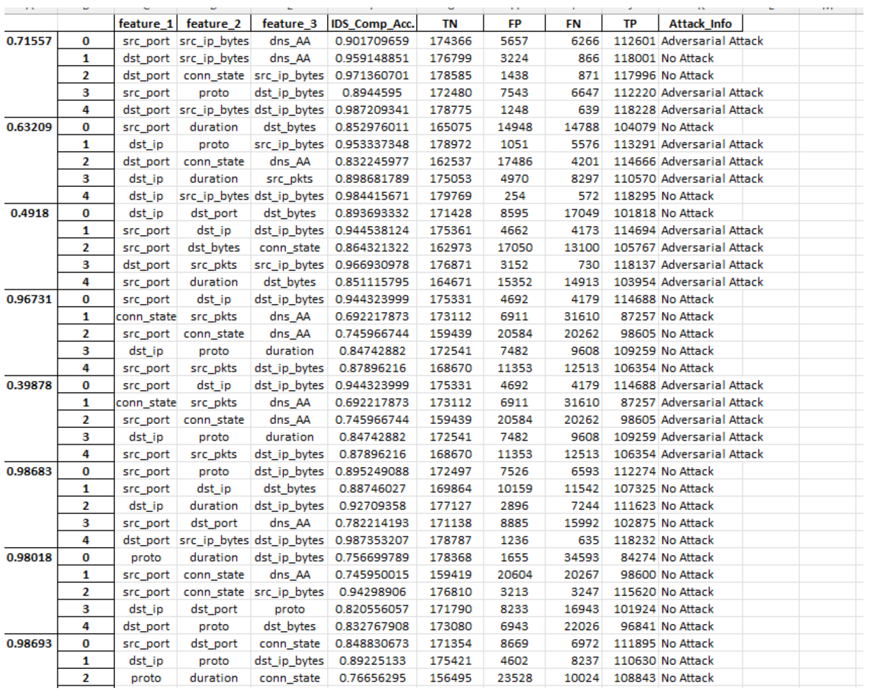


		Previous_Observation	Action	New Observation	Reward	MTD_IDS_Accuracy
1	**0**	[0. 0. 0. 0. 0.]	0	[1. 0. 0. 1. 1.]	-0.54131085	0.715574292
2	**0**	[0. 0. 0. 0. 0.]	0	[0. 1. 1. 1. 0.]	-0.641497541	0.632085383
3	**0**	[0. 0. 0. 0. 0.]	0	[0. 1. 1. 1. 1.]	-0.904920205	0.491799659
4	**0**	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.901917093	0.967305698
	1	[0. 0. 0. 0. 0.]	0	[1. 1. 1. 1. 1.]	-1	0.398782161
5	**0**	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.960503864	0.986834621
	1	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.940550035	0.980183345
	2	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.960804978	0.986934993
	3	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.959409816	0.986469939
	4	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.948469337	0.982823112
	5	[0. 0. 0. 0. 0.]	0	[1. 1. 1. 1. 0.]	-0.934355114	0.442741477
6	**0**	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.965191207	0.988397069
	1	[0. 0. 0. 0. 0.]	0	[0. 0. 0. 0. 1.]	0.565849644	0.985770685
	2	[0. 0. 0. 0. 1.]	0	[1. 1. 0. 0. 1.]	-0.625968082	0.645026598
7	**0**	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.96339456	0.987798187
	1	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.97433504	0.991445013
	2	[0. 0. 0. 0. 0.]	0	[0. 0. 0. 1. 0.]	0.556679715	0.981949881
	3	[0. 0. 0. 1. 0.]	0	[1. 1. 0. 1. 1.]	-0.884317977	0.526136706
8	**0**	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.972939878	0.990979959
	1	[0. 0. 0. 0. 0.]	0	[1. 1. 0. 0. 0.]	0.168690154	0.982605641
	2	[1. 1. 0. 0. 0.]	0	[1. 1. 1. 1. 1.]	-1	0.419522232
9	**0**	[0. 0. 0. 0. 0.]	1	[0. 0. 0. 0. 0.]	0.947877145	0.982625715
	1	[0. 0. 0. 0. 0.]	0	[1. 1. 0. 0. 0.]	0.126696778	0.959275988
	2	[1. 1. 0. 0. 0.]	0	[1. 1. 1. 0. 1.]	-0.932082706	0.446528823
10	**0**	[0. 0. 0. 0. 0.]	0	[0. 0. 1. 1. 0.]	-0.045281542	0.863732477
	1	[0. 0. 1. 1. 0.]	0	[0. 1. 1. 1. 1.]	-0.919512195	0.467479675

FIGURE 6.1: The figure above shows the first 10 episodic games played by the MTD-based IDS agent, and the corresponding reward and accuracy obtained for each step.

As seen in figure 6.1, the simulation testbed for the MTD-based IDS is set up so that the IDS agent (defending system) goes through various scenarios of which an episode could terminate, and the associated reward is obtained after an episodic termination. The attacker's incessant adversarial attacks are incorporated as part of the environment and the testbed also records in real time how the overall accuracy of the MTD-based IDS is doing for each episodic step.

Resetting to a new episode or triggering the "action" results in the retraining of the 5 IDS components (indexed $0-4$). Figure 6.2 provides a microscopic view into what goes on within an episodic step: The combination of features used to train a specific IDS component,



		feature_1	feature_2	feature_3	IDS_Comp_Acc.	TN	FP	FN	TP	Attack_Info
0.71557	0	src_port	src_ip_bytes	dns_AA	0.901709659	174366	5657	6266	112601	Adversarial Attack
	1	dst_port	src_ip_bytes	dns_AA	0.959148851	176799	3224	866	118001	No Attack
	2	dst_port	conn_state	src_ip_bytes	0.971360701	178585	1438	871	117996	No Attack
	3	src_port	proto	dst_ip_bytes	0.8944595	172480	7543	6647	112220	Adversarial Attack
	4	dst_port	src_ip_bytes	dst_ip_bytes	0.987209341	178775	1248	639	118228	Adversarial Attack
0.63209	0	src_port	duration	dst_bytes	0.852976011	165075	14948	14788	104079	No Attack
	1	dst_ip	proto	src_ip_bytes	0.953337348	178972	1051	5576	113291	Adversarial Attack
	2	dst_port	conn_state	dns_AA	0.832245977	162537	17486	4201	114666	Adversarial Attack
	3	dst_ip	duration	src_pkts	0.898681789	175053	4970	8297	110570	Adversarial Attack
	4	dst_ip	src_ip_bytes	dst_ip_bytes	0.984415671	179769	254	572	118295	No Attack
0.4918	0	dst_ip	dst_port	dst_bytes	0.893693332	171428	8595	17049	101818	No Attack
	1	src_port	dst_ip	dst_ip_bytes	0.944538124	175361	4662	4173	114694	Adversarial Attack
	2	src_port	dst_bytes	conn_state	0.864321322	162973	17050	13100	105767	Adversarial Attack
	3	dst_port	src_pkts	src_ip_bytes	0.966930978	176871	3152	730	118137	Adversarial Attack
	4	src_port	duration	dst_bytes	0.851115795	164671	15352	14913	103954	Adversarial Attack
0.96731	0	src_port	dst_ip	dst_ip_bytes	0.944323999	175331	4692	4179	114688	No Attack
	1	conn_state	src_pkts	dns_AA	0.692217873	173112	6911	31610	87257	No Attack
	2	src_port	conn_state	dns_AA	0.745966744	159439	20584	20262	98605	No Attack
	3	dst_ip	proto	duration	0.84742882	172541	7482	9608	109259	No Attack
	4	src_port	src_pkts	dst_ip_bytes	0.87896216	168670	11353	12513	106354	No Attack
0.39878	0	src_port	dst_ip	dst_ip_bytes	0.944323999	175331	4692	4179	114688	Adversarial Attack
	1	conn_state	src_pkts	dns_AA	0.692217873	173112	6911	31610	87257	Adversarial Attack
	2	src_port	conn_state	dns_AA	0.745966744	159439	20584	20262	98605	Adversarial Attack
	3	dst_ip	proto	duration	0.84742882	172541	7482	9608	109259	Adversarial Attack
	4	src_port	src_pkts	dst_ip_bytes	0.87896216	168670	11353	12513	106354	Adversarial Attack
0.98683	0	src_port	proto	dst_ip_bytes	0.895249088	172497	7526	6593	112274	No Attack
	1	src_port	dst_ip	dst_bytes	0.88746027	169864	10159	11542	107325	No Attack
	2	dst_ip	duration	dst_ip_bytes	0.92709358	177127	2896	7244	111623	No Attack
	3	src_port	dst_port	dns_AA	0.782214193	171138	8885	15992	102875	No Attack
	4	dst_port	src_ip_bytes	dst_ip_bytes	0.987353207	178787	1236	635	118232	No Attack
0.98018	0	proto	duration	dst_ip_bytes	0.756699789	178368	1655	34593	84274	No Attack
	1	src_port	conn_state	dns_AA	0.745950015	159419	20604	20267	98600	No Attack
	2	src_port	conn_state	src_ip_bytes	0.94298906	176810	3213	3247	115620	No Attack
	3	dst_ip	dst_port	proto	0.820556057	171790	8233	16943	101924	No Attack
	4	dst_port	proto	dst_bytes	0.832767908	173080	6943	22026	96841	No Attack
0.98693	0	src_port	dst_port	conn_state	0.848830673	171354	8669	6972	111895	No Attack
	1	dst_ip	proto	dst_ip_bytes	0.89225133	175421	4602	8237	110630	No Attack
	2	proto	duration	conn_state	0.76656295	156495	23528	10024	108843	No Attack

FIGURE 6.2: The figure provides a microscopic view into the first 8 episodic steps (indexed by their accuracies), which entails the feature combinations each 5 IDS components were trained with, their individual accuracies, and if they were attacked or not.

what each IDS component's accuracy is after training, and the combined IDS accuracy after training (with or without an adversarial attack).

6.2 Results and Evaluation

One of the biggest concerns of having an MTD-based system is coming up with a systematic scheme to know when to initiate an adaptation in an optimized or quazi-optimized manner. For this project, reinforcement learning was used to provide optimization and help the agent learn in real time what actions to take and when to take them to maximize profits and minimize cost. Figure 6.3 shows the cummulative reward of the MTD-based defending agent over 250 episodes. The results clearly show the agent's reward was increasing over the number episodes, which is a strong indication that learning was taking place.

Figure 6.4 shows the overall accuracy comparison of the conventional IDS and MTD-based IDS for various success rates of the attacker. The various success rates of the attacker were simulated using the python library Numpy's uniform probability distribution (using the same seed value for both the conventional and MTD-based IDS for objectivity and neutrality), and changing the odds incrementally. For example, as can be seen from the figure, at

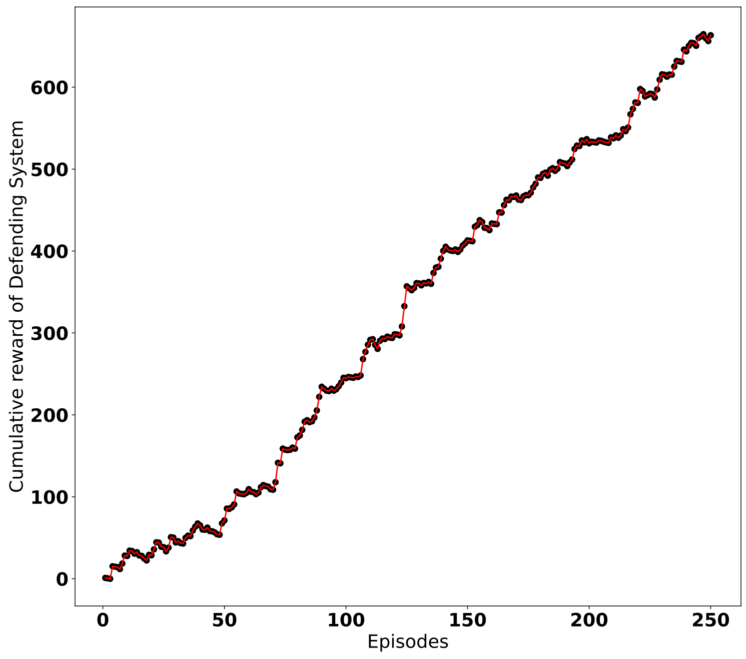


FIGURE 6.3: Graph showing the cumulative reward of the defending system (agent) during the learning process (for 250 episodes).

50 % odds of success, the attacker has an equal chance of success and failure for both the conventional and MTD-based IDS. In this case, there was no trigger for an adversarial attack, hence the reason the conventional IDS has an accuracy of close to 100 %. Neither was attacked at this success rate, however the slight drop in the accuracy of the MTD-based IDS is because of the implementation of MTD (the decentralization and aggregation MTD architecture). As the attacker's success keeps increasing however, it can be seen that the accuracy degrades significantly for conventional IDS, whereas the MTD-based IDS degrades much less comparatively.

In figure 6.5, we show the CPU usage of the MTD-based IDS in comparison to the conventional IDS for different success rates of the attacker. As can be deduced, the constraints on CPU resources increase for every increase in the adversarial attack success rate. This is more so for the MTD-based IDS because it requires shuffling of features and re-training of IDS components to mitigate attacks. It was observed that the biggest contributing factor to the high CPU usage in the MTD-based IDS is the re-training of IDS components. The conventional IDS still incurs more CPU usage because of the increase in the number of successful adversarial probing by the attacker.

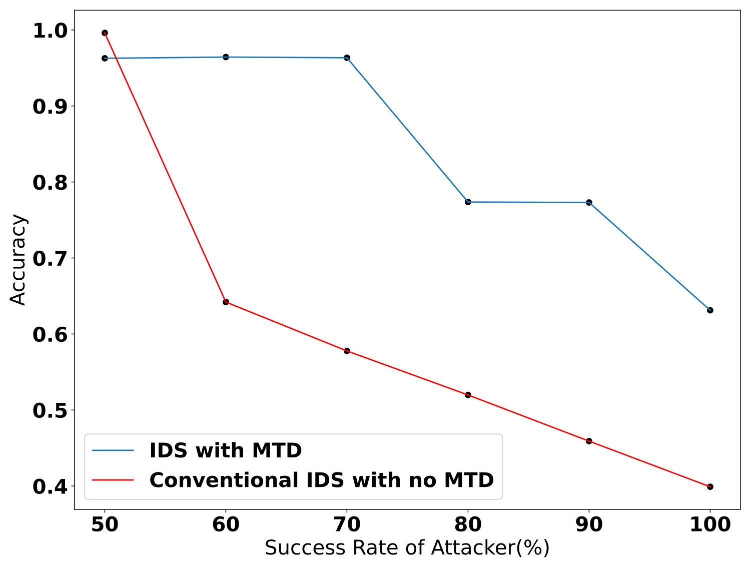


FIGURE 6.4: Graph showing the variation in accuracy for both IDS architectures, for varying success rates of the attacker.

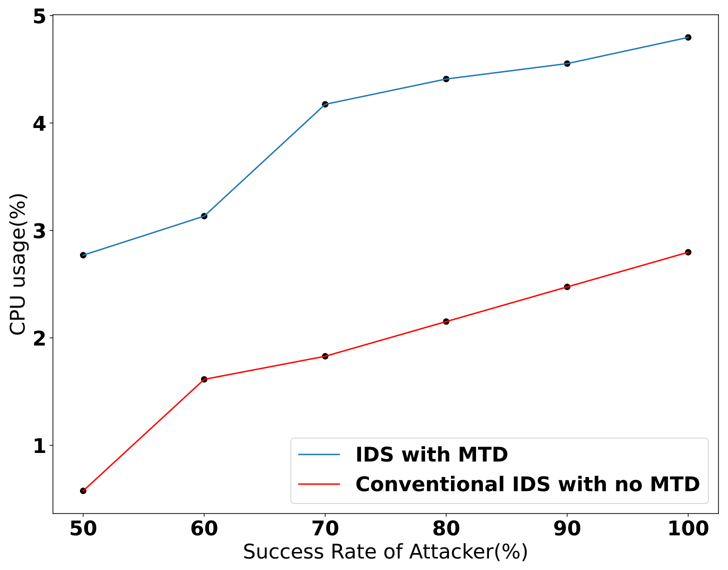


FIGURE 6.5: Graph showing the CPU usage of both the MTD-based IDS and conventional IDS for varying success rates of the attacker.

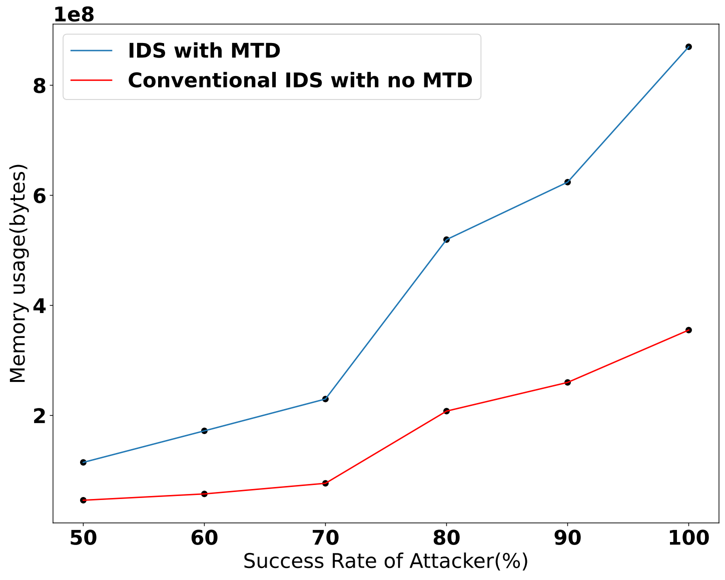


FIGURE 6.6: Graph showing the memory usage of both the MTD-based IDS and conventional IDS for varying success rates of the attacker.

Memory usage also went up for the MTD-based IDS for different attacker's success rates, as seen in Figure 6.6. This was not surprising, given the complexity of the MTD architecture that was implemented. The memory usage cost incurred by the conventional IDS as a result of more frequent adversarial activity pales in comparison to the cost incurred by the MTD-based IDS for all instances, and justifiably so.

Figure 6.7 shows a positive correlation between the reward accrued and the overall accuracy of the MTD-based IDS. This also informs us that the MTD-enabled IDS agent was truly learning to optimize and maximize its overall accuracy. It must be clarified though that the system's inherent costs were not incorporated into the reward function of the agent during the reinforcement learning phase (which is one of the limitations of the proposed model). The agent's reward was simply a function of the number of compromised IDS components and number of shuffled IDS components, as ratios of the total number of IDS components within the system.

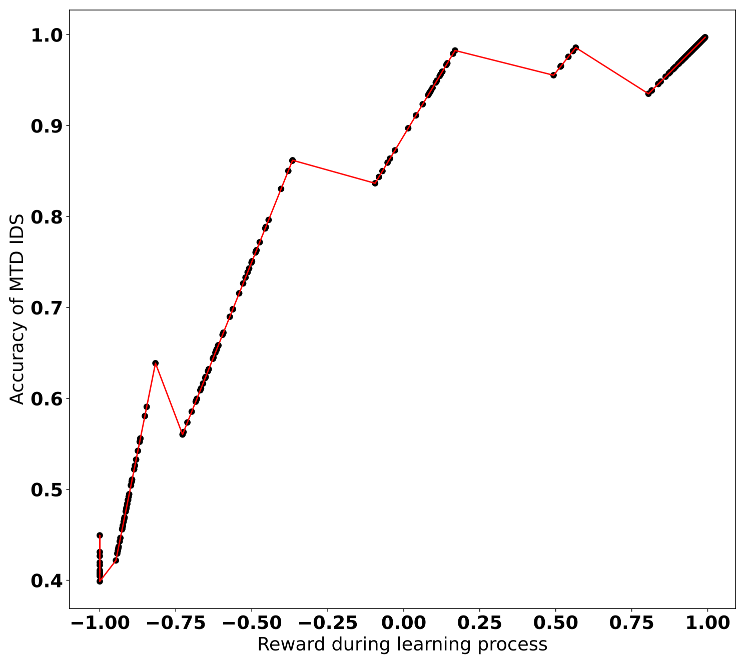


FIGURE 6.7: Graph showing the variation of the MTD-based IDS agent reward with its accuracy during the episodic game.

Chapter 7

Conclusion and Future Work

7.1 Conclusion and Discussion

The accuracy of an IDS is paramount to its functionality and performance. An IDS that is unable to accurately classify traffic within certain bounds of tolerance is deemed ineffective. We were able to implement MTD on the gateway IDS within an IoT network by proposing a stochastic game-theoretic model and then building on that to formulate the problem from a single-agent reinforcement learning perspective. The attacker's actions are incorporated into the agent's environment, and the goal is to optimize MTD adaptations. We were able to simulate the attacker's adversarial noise injection with the assumption of foreknowledge of the ToN-IoT dataset (grey box adversarial threat model), and then to compare and contrast this novel implementation with a conventional IDS under the same simulation conditions. Some of the conclusions drawn are:

- The MTD model proposed for the IDS does provide a decent mitigation wall, and cushions the IDS from the impact of an adversarial attack.
- Implementing the model comes with relatively greater costs in terms of resource consumption, compared with a conventional IDS with no MTD involved.
- Game theory and reinforcement learning are two ground-breaking mechanisms that can be used to address the MTD timing problem.
- The solution reached by the proposed model was certainly sub-optimal and could be made better by incorporating the actual costs from MTD implementation in real time during the learning process of the agent.

7.2 Future Work

The impact of the findings in this research is indisputable and will undoubtedly contribute significantly to the MTD literature. One of the drawbacks encountered was the fact that all experiments were based on simulations, hence the need to replicate the experiments within a real-world IoT environment for stronger validation. Also, the resource drainage from CPU and memory usage were not infused into the reward function of the learning agent and may have influenced the reason for the wide gaps in the curves shown. It is desired that with more research and work on the model, the gaps between the MTD-based IDS and the conventional IDS curves would become as narrow as possible.

It is also desired to expand this research further using a realistic test bed with an actual network in real time for more accurate results. This research was wholistic and dove into some of the most important topics currently within the broader research community: IoT security, Moving Target Defense (MTD), Intrusion detection systems (IDS), game theory, reinforcement learning, and adversarial attacks. The doors are open on all those research fronts, and we invite the broader IoT research community to come on board with an inquisitive scientific zeal.

Bibliography

Alotaibi, Bandar and Khaled Elleithy (2016). “A new MAC address spoofing detection technique based on random forests”. In: *Sensors* 16.3, p. 281.

Alpcan, Tansu and Tamer Basar (2006). “An intrusion detection game with limited observations”. In: *12th Int. Symp. on Dynamic Games and Applications, Sophia Antipolis, France*. Vol. 26.

Alsoufi, Muaadh A et al. (2021). “Anomaly-based intrusion detection systems in iot using deep learning: A systematic literature review”. In: *Applied sciences* 11.18, p. 8383.

Antonakakis, Manos et al. (2017). “Understanding the mirai botnet”. In: *26th USENIX security symposium (USENIX Security 17)*, pp. 1093–1110.

Ayub, Md Ahsan et al. (2020). “Model evasion attack on intrusion detection systems using adversarial machine learning”. In: *2020 54th annual conference on information sciences and systems (CISS)*. IEEE, pp. 1–6.

Booij, Tim M et al. (2021). “ToN_IoT: The role of heterogeneity and the need for standardization of features and attack types in IoT network intrusion data sets”. In: *IEEE Internet of Things Journal* 9.1, pp. 485–496.

Brunton, Steven L and J Nathan Kutz (2022). *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press.

Carlini, Nicholas and David Wagner (2017). “Towards evaluating the robustness of neural networks”. In: *2017 ieee symposium on security and privacy (sp)*. Ieee, pp. 39–57.

Ding, Guohui et al. (2018). “Game-theoretic cooperative lane changing using data-driven models”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3640–3647.

Elrawy, Mohamed Faisal, Ali Ismail Awad, and Hesham FA Hamed (2018). “Intrusion detection systems for IoT-based smart environments: a survey”. In: *Journal of Cloud Computing* 7.1, pp. 1–20.

EO, OO Ibidunmoye, BK Alese, and OS Ogundele (2013). “Modeling attacker-defender interaction as a zero-sum stochastic game”. In: *Journal of Computer Sciences and Applications* 1.2, pp. 27–32.

Evans, CDA and Joel David Hamkins (2013). “Transfinite game values in infinite chess”. In: *arXiv preprint arXiv:1302.4377*.

Feng, Ming and Hao Xu (2017). “Deep reinforecement learning based optimal defense for cyber-physical system in presence of unknown cyber-attack”. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, pp. 1–8.

Giraldo, Jairo et al. (2018). “A survey of physics-based attack detection in cyber-physical systems”. In: *ACM Computing Surveys (CSUR)* 51.4, pp. 1–36.

Giraldo, Jairo A, Mohamad El Hariri, and Masood Parvania (2022). “Moving Target Defense for Cyber–Physical Systems Using IoT-Enabled Data Replication”. In: *IEEE Internet of Things Journal* 9.15, pp. 13223–13232.

Gwon, Youngjune et al. (2013). “Competing mobile network game: Embracing antijamming and jamming strategies with reinforcement learning”. In: *2013 IEEE conference on communications and network security (CNS)*. IEEE, pp. 28–36.

Hei, Xiali et al. (2010). “Defending resource depletion attacks on implantable medical devices”. In: *2010 IEEE global telecommunications conference GLOBECOM 2010*. IEEE, pp. 1–5.

Hindy, Hanan et al. (2020). “Mqtt-iot-ids2020: Mqtt internet of things intrusion detection dataset”. In: *IEEE Dataport*.

Huang, Sandy et al. (2017). “Adversarial attacks on neural network policies”. In: *arXiv preprint arXiv:1702.02284*.

IHS, Statista (2018). “Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)”. In.

Jia, Quan, Kun Sun, and Angelos Stavrou (2013). “Motag: Moving target defense against internet denial of service attacks”. In: *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*. IEEE, pp. 1–9.

Lee, Hyungyu, Ho Bae, and Sungroh Yoon (2020). “Gradient masking of label smoothing in adversarial robustness”. In: *IEEE Access* 9, pp. 6453–6464.

Li, Yuxi (2017). “Deep reinforcement learning: An overview”. In: *arXiv preprint arXiv:1701.07274*.

Liu, Yu, Cristina Comaniciu, and Hong Man (2006). “A Bayesian game approach for intrusion detection in wireless ad hoc networks”. In: *Proceeding from the 2006 workshop on Game theory for communications and networks*, 4–es.

Mahmood, Kaleel et al. (2021). “Beware the black-box: On the robustness of recent defenses to adversarial examples”. In: *Entropy* 23.10, p. 1359.

Makhdoom, Imran et al. (2018). “Anatomy of Threats to the Internet of Things”. In: *IEEE Communications Surveys & Tutorials* 21.2, pp. 1636–1675.

Maschler, Michael, Shmuel Zamir, and Eilon Solan (2020). *Game theory*. Cambridge University Press.

Mercado-Velázquez, Andrés Aharhel, Ponciano Jorge Escamilla-Ambrosio, and Floriberto Ortiz-Rodriguez (2021). “A moving target defense strategy for internet of things cybersecurity”. In: *IEEE Access* 9, pp. 118406–118418.

Navas, Renzo E et al. (2020). “MTD, where art thou? A systematic review of moving target defense techniques for IoT”. In: *IEEE internet of things journal* 8.10, pp. 7818–7832.

Osei, Arnold Brendan et al. (2022). “Optimized Moving Target Defense Against DDoS Attacks in IoT Networks: When to Adapt?” In: *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, pp. 2782–2787.

Roy, Sankardas et al. (2010). “A survey of game theory as applied to network security”. In: *2010 43rd Hawaii International Conference on System Sciences*. IEEE, pp. 1–10.

Rubio, Juan E, Cristina Alcaraz, and Javier Lopez (2017). “Preventing advanced persistent threats in complex control networks”. In: *European Symposium on Research in Computer Security*. Springer, pp. 402–418.

Santhanam, Gokula Krishnan and Paulina Grnarova (2018). “Defending against adversarial attacks by leveraging an entire GAN”. In: *arXiv preprint arXiv:1805.10652*.

Santos, Leonel, Carlos Rabadao, and Ramiro Gonçalves (2018). “Intrusion detection systems in Internet of Things: A literature review”. In: *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, pp. 1–7.

Şendroiu, Adrian and Vladimir Diaconescu (2018). “Hide’n’seek: an adaptive peer-to-peer iot botnet”. In: *architecture* 3, p. 5.

Stolfo, Salvatore J et al. (2000). “Cost-based modeling for fraud and intrusion detection: Results from the JAM project”. In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX’00*. Vol. 2. IEEE, pp. 130–144.

Tavallaee, Mahbod et al. (2009). “A detailed analysis of the KDD CUP 99 data set”. In: *2009 IEEE symposium on computational intelligence for security and defense applications*. Ieee, pp. 1–6.

Von Neumann, John and Oskar Morgenstern (2007). “Theory of games and economic behavior”. In: *Theory of games and economic behavior*. Princeton university press.

Wang, Huangxin, Fei Li, and Songqing Chen (2016). “Towards cost-effective moving target defense against ddos and covert channel attacks”. In: *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pp. 15–25.

Wang, Li and Dinghao Wu (2016). “Moving target defense against network reconnaissance with software defined networking”. In: *Information Security: 19th International Conference, ISC 2016, Honolulu, HI, USA, September 3-6, 2016. Proceedings 19*. Springer, pp. 203–217.

Xu, Han et al. (2020). “Adversarial attacks and defenses in images, graphs and text: A review”. In: *International Journal of Automation and Computing* 17, pp. 151–178.

Zaman, Shakila et al. (2021). “Security threats and artificial intelligence based countermeasures for internet of things networks: a comprehensive survey”. In: *Ieee Access* 9, pp. 94668–94690.

Zantedeschi, Valentina, Maria-Irina Nicolae, and Ambrish Rawat (2017). “Efficient defenses against adversarial attacks”. In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 39–49.

Zhang, Haichao and Jianyu Wang (2019). “Defense against adversarial attacks using feature scattering-based adversarial training”. In: *Advances in Neural Information Processing Systems* 32.

Zhuang, Rui, Scott A DeLoach, and Xinming Ou (2014). “Towards a theory of moving target defense”. In: *Proceedings of the first ACM workshop on moving target defense*, pp. 31–40.