An Improved Predictive Model for Early Detection of Advanced Persistent Threat Attacks on High Value Networks

by Weina Feng Dorsky

B.S. in Electrical Engineering, Jun 2003, Boston University
M.S. in Electrical Engineering, Jun 2006, University of Rhode Island
M.B.A, May 2009, Babson University

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

May 19, 2024

Praxis directed by

John Fossaceca
Professorial Lecturer of Engineering Management and Systems Engineering

Shahryar Sarkani
Adjunct Professor of Engineering Management and Systems Engineering

The School of Engineering and Applied Science of The George Washington University certifies that Weina Feng Dorsky has passed the Final Examination for the degree of Doctor of Engineering as of November 30, 2023. This is the final and approved form of the Praxis.

An Improved Predictive Model for Early Detection of Advanced Persistent Threat Attacks on High Value Networks

Weina Feng Dorsky

Praxis Research Committee:

John Fossaceca, Professorial Lecturer of Engineering Management and Systems Engineering, Praxis Director

Shahryar Sarkani, Adjunct Professor of Engineering Management and Systems Engineering, Committee Member

Muhammad Islam, Professorial Lecturer of Engineering Management and Systems Engineering, Committee Member

Dedication

The author wishes to thank her husband, Jason Dorsky, for being so supportive in taking care of the kids and providing technical writing review and recommendations for the completion of this praxis. Also, she wishes to thank her parents for helping out with childcare and family chores as needed during these two long years. She would like to thank her children, Miles and Jameson, for always being there for her, both mentally and physically, when she has needed them the most.

Acknowledgements

The author wishes to thank Dr. Fossececa and Dr. Sarkani for the continued support and constructive feedbacks and motivation in getting this Praxis completed. The author also has grand appreciation to Dr Branka Stojanovic @ Joanneum Research for providing the semi-synthetic dataset for validating the Praxis. The author thanks her husband the technical writer who helps improve the quality of the praxis.

Abstract of Praxis

An Improved Predictive Model for Early Detection of Advanced Persistent Threat Attacks on High Value Networks

Hackers have stolen enormous amounts of intellectual property data from high value networks, resulting in $300 billion of intellectual property stolen each year. Cyberattacks have increased significantly in the past several years with the exponential growth in internet usage and cloud-based storage. Of these attacks, the most impactful is the Advanced Persistent Threat (APT).. The praxis proposes a 2-stage classification/detection model to improve the detection of APT attacks on high value networks using K Nearest Neighbor (KNN) —a supervised matching learning (ML) algorithm for the first stage classification, followed by an unsupervised ML model—the isolation forest for the second stage detection. This praxis uses a more realistic dataset representing actual network traffic with both benign and threat information—CICIDS2017 in Packet Capture (PCAP) files which contain packet data including various protocols, attack vectors, IP addresses of source and destination, and port numbers. In addition, the CICIDS2017 is augmented with Contagio's real-time APT datasets to create a near real-time true APT embedded network traffic dataset. Contagio is a collection of historic malware samples, observations, threats and analysis that's downloadable on the contagion website. The semi-synthetic dataset enables more realistic performance measurements from the various ML / Deep Learning (DL) algorithms. The performance metrics collected from the model show improved detection of anomalies and APT attacks. The improvement in accuracy and precision for the anomaly detection range from 5% up to 40%.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CICFLOWMETER	A network traffic flow generator distributed by CIC to generate 84 network traffic features from GITHUB.
CSV	Comma – separated values
DBN	Deep belief network
DL	Deep Learning
KNN	K-nearest neighbors
ML	Machine Learning
PCAP	Packet Capture
SVM	Support Vector Machine
APT	Advanced Persistent Threat
CONTAGIO	APT dataset
CSV	Comma Separated value
FTP	File Transfer Protocol
SSH	Secure shell
XSS	Cross-site scripting
CPU	Control Processing Unit

IRC	Internet Relay Chat
SNMP	Simple Network Management Protocol
C&C	Command and Control
SMO	Social Media Optimization
FPR	False Positive Rate
ROC	Receiving Operating Characteristics
AUC	Area Under the Curve
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
GAN	Generative Adversarial Network
RBN	Restricted Boltzmann Machine
ANN	Artificial Neural Network
NB	Naïve Bayes
HMM	Hidden Markov Model

Chapter 1—Introduction

1.1 Background

According to Check Point Research ("Informa PLC Informa UK Limited," 2023), in 2022 cyberattacks increased globally by 38%, where the U.S. alone saw a 57% increase in overall attacks. These attacks occurred primarily in the Education/Research, Government and Healthcare industries,[1] highlighting the needs for better cyber threat detection mechanisms. The enterprise network infrastructure of these industries (high value networks) generates huge amounts of private, controlled unclassified and business sensitive data. Protecting this data and associated network infrastructures hosting the data has become critical to conserve companies' competitive advantages. Cybercriminals have evolved their weaponry, using sophisticated attack methods that implement various techniques such as phishing, hacking, ransomware, installation and execution of malicious code, and penetrating internal networks. One form of cyberattack that poses a particularly significant threat is called Advanced Persistent Threats (APT), a stealthy and continuous attack campaign intended to steal data from a company or network undetected (Khaleefa & Abdulah, 2022).

Early APT attacks were typically confined to nation-states targeting government resources, but they have since expanded into private and business sectors. According to Khaleefa and Abdulah (Khaleefa & Abdulah, 2022), these attacks have become "a form of continuous hacking and a set of covert approaches targeted against a particular institution with high-value data, such as the government military or financial industry." Current security solutions have rapidly become outpaced by the growing complexity and sophistication of these APT attacks, making it hard for businesses and many different

organizations to defend themselves and safeguard their sensitive information and resources. APT attacks are typically implemented in multiple steps, where network systems with known or undiscovered zero-day vulnerabilities are targeted and exploited (Neuschmied et al, 2022). As a result, these attacks are typically hard to detect in real time.

In an effort to combat APT attacks, intrusion detection systems (IDS) have been designed to recognize and detect hostile behavior patterns in current networks, and have become an important tool for detecting complex network attacks (Maseer et al, 2021). However, even after many years of development, IDS still face challenges in their detection accuracy, ability to detect unknown attacks, and ability to reduce false alarm rates (Liu & Lang, 2019). Researchers have implemented both ML/DL in attempts to improve IDS algorithms. ML can detect differences between normal and abnormal data with high accuracy, and is an effective method for detecting unknown attacks. DL is a subset of ML that can learn features automatically, and has become increasingly leveraged (Liu & Lang, 2019). Traditional ML and DL also differ based on their data and hardware (HW) dependencies, execution time and interpretability (Xin et al, 2018). One major challenge in addressing APT is the limited availability of representative APT datasets that can be applied in research. Previous efforts frequently used NSL-KDD datasets which are over 20 years old with simulated TCP/IP network traffic and malware (such as Distributed Denial of Service (DDOS) and bootstrap) embedded in it, which are now outdated and not accurate samples of network data. In this Praxis, a semi-synthetic dataset will be used to mimic real-world network traffic with real-time APT malware data. Various ML and DL algorithms will also be reviewed to determine which perform

better and faster within the APT representative dataset created with both CICIDS2017 and Contagio Malware datasets.

1.2 Research Motivation

In recent years, the Internet has become an essential necessity of everyone's life, used for both professional and personal purposes. In nearly all use cases, sensitive data that individuals wish to protect are accessible via the Internet, which results in unavoidable exposure to threats and malware attacks. Regardless of the industry, most large companies have invested significant resources to securely store sensitive, private and valuable company data, using local databases or storage with one of the major cloud service providers. Government organizations, including the Department of Defense (DoD) and Defense Industrial Base (DIB), have done the same. Malicious entities have adapted to these protection efforts and have conducted APT attacks against defense contractors' networks, exploiting controlled unclassified information without alerting the host or users for significant periods of time (Devore & Lee, 2017). In fact, many of these companies do not notice the attack until one to two years later, at which time they have already incurred significant damage. Nation state actors not only seek to steal defense information, but also target financial institutions, entertainment corporations and health organizations. These attacks may originate from nation states, but also may be conducted by individual cyber actors. The scope of APT attacks and targeted entities has continued expanding in recent years with advancing technology. The motivation for this research is to address the growing and persistent threat of APT through review of various ML/DL algorithms that can help detect attacks early to prevent and mitigate the amount of high value data exploited.

1.3 Problem Statement

> *Software hackers from nation states such as China have stolen enormous amounts of intellectual property data from high value networks, resulting in $300 billion of intellectual property stolen each year.*
>
> Brown & Pavneet, 2018

According to Brown and Pavneet (Brown & Pavneet, 2018), China is the most aggressive country targeting the U.S. using industrial espionage, with hundreds of thousands of Chinese army professionals deployed to carry out cyber theft. This is occurring in academia and in the defense industry as China increasingly controls the supply chain of U.S. military equipment and services. In 2012, Former NSA Director General Keith Alexander told Congress that "the greatest transfer of wealth in history," was U.S. companies' loss of "about $250 billion per year" through intellectual property theft, with another "$114 billion due to cyber-crime," which he termed "our future disappearing in front of us." Furthermore, the IP Commission Report of 2013—a study led by top officials (leading Americans from the private sector, public service in national security, foreign affairs, academia, and policies) states that 96% of the world's cyber espionage originated from China, with an estimated $100 billion in lost sales and $300 billion in stolen IP occurring each year. Some prominent examples of this theft include an attack on the U.S. Office of Personnel Management (OPM), where 4.2 million government employee personnel files were stolen, as well as over 21.5 million individuals' clearance background reports, and the exfiltration by People's Liberation Army of China (PLA) Unit 61398—a covert cyberforce within the PLA of China—which has stolen significant amount of data concerning technologies in aerospace, satellite design, telecommunications and IT from companies that includes Google, Adobe and others. This cyber theft and industrial espionage will expedite the illicit technology

transfer to China and put U.S. at risk of losing technological superiority and battlefield dominance.

1.4 Thesis Statement

A two-stage classification/detection model is required to improve the detection of APT attacks on high value networks.

In recent years, many researchers have reviewed various machine learning (ML) and deep learning (DL) algorithms for anomaly-based intrusion detection systems (Maseer et al, 2021). These include Artificial Neural Network (ANN), Decision Tree (DT), K-nearest neighbor (K-NN), Naïve Bayes (NB), Random Forest (RF), Support Vector Machine (SVM), Convolutional Neural Network (CNN) and others. This praxis reviews various intrusion detection systems and proposes a novel approach: a classification and detection model that can help improve the detection of APT attacks early to prevent further lateral movement within targeted networks.

1.5 Research Objectives

- **RO1:** To employ a more real-time representative network dataset with real world APT attack in development of an APT classification model.
- **RO2:** To identify the significant features in detection of APT attack on high value networks.
- **RO3:** To develop a predictive model to detect APT attacks on high value networks.
- **RO4:** To identify which ML algorithm is best for detection of APT attacks on high value networks.

1.6 Research Questions and Hypotheses

This research investigates early detection of APT attacks. To achieve the research objectives above, the research questions below will be studied in this praxis

RQ1: What are significant features in detection of APT attack on high value networks?

APT are known to hide in weak signals and large amounts of data and remain hidden over a long period of time to avoid detection. Most of the research in this area provides limited basic information on the features used for APT detection (Hofer-Schmitz et al, 2021). This praxis plans to investigate the impacts of statistical network traffic features captured in the PCAP files, and identify any significant features of APT attacks.

RQ2: Can a classification model be developed to detect APT attacks on high value networks?

For the purposes of this Praxis, a classification model is defined as a model that can detect APT attacks directed at targeted corporate enterprise networks. Many researchers identify machine learning as one of the promising solutions for detecting APT attacks (Myneni et al, 2020).

RQ3: Which ML algorithm is best for classification of APT attacks on high value networks?

Within the various learning techniques in ML/DL, can we identify which performs the best for the semi-synthetic dataset? (Khaleefa, 2022) The below figure classifies known ML/DL Algorithms.

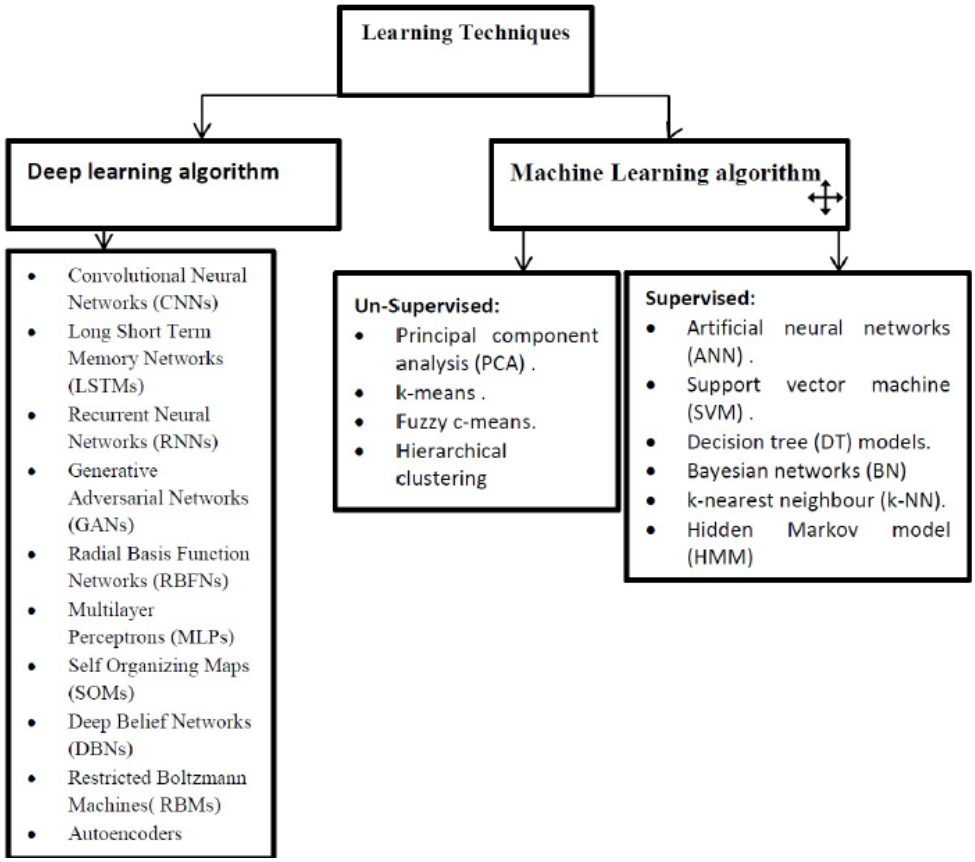


Figure 1: Classification of Known ML and DL Algorithms (Khaleefa, 2022)

Below are the hypotheses created from the research questions:

H1: Destination information and packets information (forward and backward) are significant features in detection of APT attacks on high value networks.

H2: A classification/detection model can detect APT attacks on high value networks with 90% accuracy.

H3: A combination of the Isolation Forest and KNN provides the best accuracy and precision for APT detection.

1.7 Scope of Research

APT has become a threat to many national governments and organizations, and can more easily evade existing security tools (Khaleefa, 2022). The scope of this research is directed to determine improved approaches for detecting APT attacks on enterprise networks through analysis of ML/DL algorithms using a semi-synthetic real-time dataset. In particular, the research will analyze the performance of the above-referenced ML/DL algorithms (see figure 1) against the semi-synthetic dataset created by Dr Hofer-Schmitz and Dr Stojanovic. Some performance measurements include true positive and negative rates, accuracy, precision, recall and F1-Score. The training and testing time is also an evaluation factor that will be reviewed. The dataset used will be limited to the semi-synthetic dataset of a combination of the CICIDS2017 and Contagio APT data files.

1.8 Research Limitations

One significant research limitation of this praxis is the lack of true real-time APT datasets. Most companies that have been the subject of an APT attack are not willing or able to disclose information about the attack, or not willing to share the network flow data captured from their enterprise network infrastructure's logs. This network data could provide significant benefits in designing protections against APT attacks, but due to its sensitive nature, exposing this information could put companies at risk of future exploitations.

In addition, the semi-synthetic data used for this praxis has certain drawbacks. With regular synthetic data, there is possibility of failure of detection in realistic condition. With semi-synthetic data, a biased dataset is introduced, and it may not sufficiently reflect accurate synthetic user modeling. Advantages of using this semi-synthetic data is ease of analysis with well understood data as well as proper features; in addition, the dataset provides a much less costly solution with better scalability and adaptability (Stojanovic et al, 2020),

1.9 Organization of Praxis

The praxis is organized into five sections. Chapter One includes the background of the praxis as well as the problem statement, the thesis and a list of research questions. Chapter Two provides a literature review of APT attacks as well as research on ML/DL for intrusion detection of APT attacks. Chapter Two also identifies gaps in research and contributions to the body of knowledge. Chapter Three addresses the data and methodologies used to test the research hypotheses. Chapter Four reviews the results from hypotheses testing, and Chapter Five will conclude with a summary of the research and recommendations for future work.

Chapter 2—Literature Review

2.1 Introduction

Advanced Persistent Threat (APT) is often sponsored by nation states that target specific companies, industries, and military branches in an effort to steal sensitive information, trade secrets, and military designs, with the goal of illicitly obtaining advancements in technology and economic advantages. To combat against APT, early detection is critical. In the below sections, the characteristics of APT, the lifecycle of APT, the types of APT and various detection and prevention methods are described.

2.1.1 APT Characteristics

The amount of research concerning APT has significantly increased in recent years, particularly in response to the rising number, complexity, and geographical expansion of such attacks, with recent attacks spreading throughout many countries in Asia, North America and Europe (Lemay, et al, 2018). Recent surveys on APT attacks (Alshamrani et al., 2019; Khaleefa et al., 2022; Stojanovic et al., 2020; Ibrahim et al., 2018) and related techniques, solutions and challenges for addressing such attacks typically begin by addressing what it means for an attack to be an "Advanced Persistent Threat." These individual facets of this method of attack are summarized below.

Advanced: The attacker boasts technical skills and uses several threat vectors, including malware, vulnerability scanning, spear phishing and social engineering to exploit weaknesses inside the target (Stojanovic et al., 2020). The APT attackers are usually well-funded by potential nation states, who has unlimited budget with access to advanced tools. The advanced methods ensure that the attacks have been launched and are maintained over time (Alshamrani et al., 2019).

Persistent: APTs frequently occur in stages and persist over an extended period of time. These stages include identifying an organization's vulnerabilities, exploiting those vulnerabilities via multiple threat vectors, expanding control once access is gained and continuing the attack, which may include long-term planning considerations to avoid detection (Stojanovic et al., 2020). The attackers also do not give up; once the attackers gain access into the targeted system, they try to remain in the system for as long as possible without being detected. In order to accomplish this, the attackers typically use several evasive techniques to elude detection by their target's intrusion detection system. A so-called "low and slow" approach is often used to increase the rate of the success of the attack (Alshamrani et al., 2019).

Threat: The attacker is typically motivated and capable of successfully carrying out the attack. The objective is to obtain assets, money, intellectual property (IP) and confidential data from designated organizations such as government agencies, critical infrastructure systems and financial institutions (Stojanovic et al., 2020). The threat can lead to the exposure of any sensitive data or the impediment of critical components or missions. These are rising threats to many national entities and organizations that have established advanced protection systems in an effort to guard their mission and data (Alshmarani et al., 2019).

2.1.2 Attack Process

The National Institute of Standards and Technology (NIST) defines APT in its 800-39 Managing Information Security Risk publication as an attack that "(i) pursues its objectives repeatedly over an extended period of time; (ii) adapts to defenders' efforts to resist it; and (iii) is determined to maintain the level of interaction needed to execute its

objectives." To achieve these goals, the attackers may attempt to completely bring down or disable the business they target, or, alternatively, they may attempt to slowly infiltrate a network and siphon data without being noticed. As pointed out by Auty (Auty, 2015), for a business, the hardest part of combating these types of APT attacks is the difficulty, to detect the attacks even after installing various powerful and useful intrusion detection and prevention systems (IDS/IPS). Various papers illustrate an APT threat "life cycle" (or APT "attack model") that implements multiple stages of attack, which can increase the attack's evasion and success rate (Ibrahim et al., 2018; Quintero-Bonilla et al., 2020). Quintero-Bonilla's paper describes APT threat life cycles that can range from 3 to 11 steps (see below summary table – Table 1 from the paper).

Likewise, Mandiant's report (a company that provides early threat insights through intelligence and response expertise for the highest-profile incidents) presents an APT life cycle of 8 stages: "1) initial recon, 2) initial compromise, 3) establish a foothold, 4) escalate privileges, 5) internal recon, 6) move laterally 7) maintain a presence, and 8) complete mission after completing its APT1 analysis" (Mandiant, 2013). These various multi-step models are very similar in context and operation. Many of the recent APT literature reviewed has depicted a more generalized model (see Figure 2).

3 Stages [68]	4 Stages [7]	4 Stages [61]	5 Stages [9]	5 Stages [69]	6 Stages [70]	6 Stages [2]	7 Stages [72]	7 Stages [73]	8 Stages [14]	11 Stages [1]
Initial Compromise	Information Collection	Initial Compromise	Reconnaissance	Delivery	Intelligence gathering	Reconnaissance and weaponization	Research	Reconnaissance	Initial recon	Initial access
	Intrusion phase		Incursion		Initial Compromise	Delivery	Preparation	Weaponization	Initial compromise	Persistence
						Initial intrusion	Intrusion	Delivery		Privilege Escalation
Lateral movement	Lateral expansion	C&C	Discovery	Exploit	C&C	C&C	Conquering network	Exploitation	Establish foothold	Discovery
		Lateral movement	Capture	Installation	Lateral movement Assets/Data discovery	Lateral movement	Hiding presence	Installation	Escalate privileges Internal recon	Lateral movement
									Move laterally	
Command and control	Information theft phase	Attack achievement	Ex-filtration	C&C	Data ex-filtration	Data ex-filtration	Gathering data	C&C	Maintain presence	Collection
				Actions			Maintaining access	Actions on objective	Complete mission	Exfiltration
										Stages executed in parallel: Execution, Defence evasion, Credential access, and Command & control

Table 1: Various APT Life Cycles
(Credit Table 4 of Quintero-Bonilla et al., 2020)

Figure 2: Simplified APT Life Cycle

The steps in this simplified model are summarized below:

Stage 1: Reconnaissance – This stage describes the attacker's attempts to understand the target and learn and collect information on everything from the structure of the target organization, to the type of network infrastructure implemented, to any public or non-public information of value. The more information the attacker can gather, the higher the success rate of reconnaissance. Most reconnaissance efforts are conducted using spear-phishing techniques (e.g., via an attachment in an email or a link to a compromised server), social engineering (e.g., to obtain passwords), open-source intelligence, watering holes and port scanning.

Stage 2: Establish Foothold – In this stage, the attacker has made several attempts to penetrate the target network, and at this point has gained access to the computer/server and established a connection to the command and control (C&C) server. Here, the attacker can likely utilize different tools such as backdoors, remote access tools, unauthorized applications, network scans/exploitation to control the target system. A zero-day vulnerability is sometimes used to identify any network holes that can be used for intrusion.

Stage 3: Maintain Presence – In this stage, the attacker is trying to remain hidden and maintain a stealth presence for a long time. The attack may have periods of inactivity to ensure continuous control over key systems. For example, the attacker may use legitimate VPN credentials and/or login through web portals using stolen accounts to maintain a presence.

Stage 4: Lateral Movement – In this stage, the APT attack spreads across the network to additional servers and attempts to escalate its privilege to obtain more sensitive and private data, particularly if the attacker's goal is to steal an organization's critical data. The attacker typically utilizes standard operating systems which makes it difficult to detect their actions. (Stojanovic et al., 2020)

Stage 5: Exfiltration – In this final stage, mission success requires delivery of the accumulated data to an external attacker's C&C server or cloud. This can be done through a burst of information exfiltration or through the gradual exfiltration carried out without the target's knowledge. This stage is where critical data can be destroyed as well, causing significant (and possibly irreparable) harm.

APT attacks have expanded beyond targeting government entities and now target private and corporate sectors with special purposes and objectives. The below table (Table 2) summarizes differences between APT attacks versus traditional attacks.

Feature	APT Attack	Common/Traditional Attack
Attacker	Organized criminal group / Government sponsored nation states	Single person / a hacker

Feature	APT Attack	Common/Traditional Attack
Target	Specific organizations, government institutions, technology companies, the health industry, financial sectors, etc.	Individual system / person
Purpose	Stealing confidential information, IP data, sensitive information; Obtain strategic and competitive advantages	Financial benefits, show-offs
Attack life cycle	Low and slow – maintain stealth	Single occurrence or short period of time

Table 2: APT Attacks and Traditional Attacks
(Quintero-Bonilla et al., 2020; Khaleefa, et al, 2022)

Organizations can use the information above to assess if a detected attack is an intentional APT attack. Three criteria can be used to carry out this determination.

1. Can the attack be prevented? – If minimal countermeasures and security procedures can help prevent a possible but unexpected attack from occurring, that attack is likely not an APT attack.
2. Does the attack require reconnaissance? – If the attack does not require any significant assessment of the target environment and does not require time to study the defensive environment and implemented security measures, it is likely not an APT attack.
3. Is the attack readily identifiable? – If the technique and approach of the attack is not novel, is not unique or does not implement various techniques, such that the attack can be easily detected, it is likely not an APT attack.

2.1.3 Classic APT cases

Mandiant publishes many of the APT attack cases ranging from APT1 – APT41 (Mandiant, 2013). Below is a summary of all the APT cases (Table 3).

Year	APT	Attack Organization	Target	Overview
2006	APT1 (unit 61398)	China People's Liberation Army (PLA)	IT, Aerospace, SATCOM, Scientific Research, Energy, Transportation, Construction, Navigation, Financial Services, Advertising, Healthcare, & Education	Has stolen hundreds of terabytes of data from many organizations, (Over 140 at least), most of these companies are across a broad range of industries in English-speaking countries. Via spear phishing with malicious hyperlinks.
2007, 2009, 2015, 2010, Since 2012	APT2 – APT27, APT30, APT31, APT40, APT41	China Espionage groups; Freelance group	Regional telecom providers; Asia-based employees of global Telecom; Military application technology in the US, Europe, and Asia; Media and entertainment, aerospace and defense in the US, Germany UK, India, Japan, and Taiwan. Global financial, trade, energy, economic, and military sectors in the US, European countries, and South Africa; Government in Taiwan and Japan; Organizations headquartered in multiple countries under healthcare, defense and aerospace	IP theft via Moose and Warp with spear phishing email; Espionage group targeting to acquire military and intelligence info to gain Chinese national security goals. Using browser-based exploits such as zero-day (IE, Firefox and Adobe Flash) and malware with keylogging capabilities. IP theft in U.S. and U.K. with infiltration from one organization to another and lateral movement. Spear Phishing data theft of biotechnology and pharmaceutical industry.
2016	APT28	Russian Government	Collecting intelligence on defense and geopolitical issues	SOURFACE downloader, and backdoor EVILTOSS and implanted CHOPSTICK. Employed RSA encryption to protect stolen files and information

Year	APT	Attack Organization	Target	Overview
	APT32	Vietnam (Oceanlotus group)	Foreign companies investing in Vietnam's manufacturing, consumer products, consulting and hospitality sectors	Erode competitive advantages of targeted organization. Social engineering tactics were used.
2014	APT33 APT34	Iran	Various industries, including but not limited to government, financial, chemical, energy, and telecommunication – Middle East US, & Western Europe; Middle Eastern military and diplomatic personnel.	Reconnaissance effort to benefit Iranian nation-state interested – leveraged Microsoft office vulnerability to deploy POWRUNER and BONDUPDATER. Spear phishing and social engineering
	APT38	North Korea (Scarcruft & Group123)	South Korea mainly, but also in the Asian countries such as also Japan, Vietnam and the Middle East – a variety of industry including electronics, aerospace, automotive, chemical, manufacturing, and healthcare.	Expanding in scope and sophistication, using zero-day (adobe flash) and wiper malware and social engineering.

Table 3 Summary of Classic APT Cases (https://www.mandiant.com/resources/insights/apt-groups)

According to the Mandiant data above, notorious nation states such as China, North Korea, Russia and Iran have been conducting APT attacks since 2006. Significant APT attacks include the American/Israeli Stuxnet worm that was famously used to take control of the supervisory control and data acquisition systems (SCADA) to disrupt Iran's nuclear program; the Flame Virus that was also used to disrupt Iranian uranium enrichment capabilities; China's Shady Rat operation that stole data from foreign defense contractors; and China's recent attack on the U.S. OPM where 22.1 million records containing personally identifiable information were exfiltrated (Devore et al., 2017). All

of these APT attacks have had significant detrimental impacts on various governments and organizations.

Techniques mostly used by the nation states are described below:

Social Engineering: this Tactics, Techniques and Procedures (TTP) manipulates individuals to compromise valuable and sensitive data. This tactic is a challenge to the networks regardless of the firewalls, cryptography methods, antivirus or intrusion detection systems. Social engineering is the most powerful attack because they cannot be prevented using software (SW) or HW solutions. (Salahdine et al., 2019)

Spear Phishing: this TTP utilizes mass-mail phishing campaign, spammed out to thousands of users in the hopes that some take the bait. They target specific organizations, duping individuals to download malware onto their machine. With a 19% success rate, breaches in Google and RSA demonstrate that this technique can evade detection by traditional security tools. (Parmar, 2012)

Watering Hole: this TTP targets access to organizations' user credentials, financial information and other secret information by infecting websites that the members of the organization are known to visit. (Ismail et al., 2017)

Drive by downloads: this TTP intends to make a user unintentionally download and execute malicious SW. The malware is downloaded by exploiting security weakness, browser exploits or embedded plug-ins such as adobe flash, java/javascript or activeX. (Khaleefa et al., 2022)

Zero-day: exploits where the vulnerability has not been disclosed publicly. The new vulnerability helps the attacker go unnoticed. The attack information is not available until an attack is discovered. (Bilge et al., 2012)

In the next section, the detection and prevention of APT will be discussed.

2.2 Detection and Prevention of APT

As discussed in the introduction, APT attacks are difficult to detect and there are currently no specific intrusion detection methods, anti-virus software or other single tool that can successfully and reliably detect such attacks. Yang et al. modeled a network subjected to an APT attack and proposed defense strategies to the security metrics of equilibrium security: "A new security metric of cyber networks, is defined as the expected fraction of the compromised nodes in the equilibrium." (Yang et al., 2017) Several factors that impact equilibrium security via theoretical analysis and simulation contribute to defensive solutions against APT. For example, external attack and internal infection are two known threats of APT attacks. There are also two types of defensive measures: prevention and recovery. The equilibrium security is based on these two premises (prevention and recovery); both depend on increasing resources per unit of time used for attacking a node. In summary, Yang suggests that investing more resources into configuring the defense of the network is more effective than enhancing security for APT attacks. According to Alshmarani's paper, there are three defensive APT methods to consider: Monitoring, Detection and Mitigation. (Alshmarani et al., 2019).

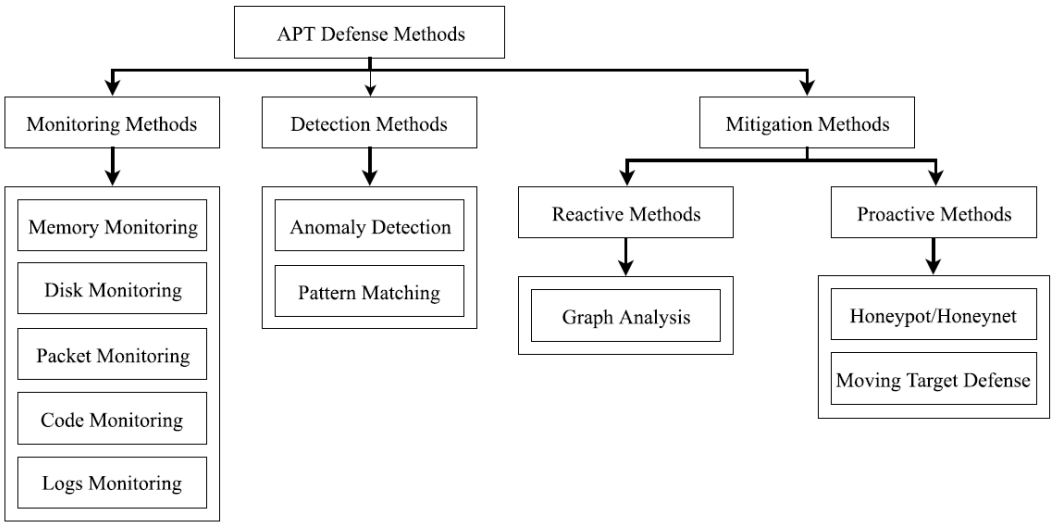


Figure 3: APT Defense Method Classification (Alshmarani et al., 2019, figure 8)

2.2.1 Monitoring methods

Effective detection of any attacks, traditional or APT, requires monitoring methods at each node, each entry point, each network layer and throughout the entire network.

1. Memory Monitoring: Using this method, memory use is tracked, and any unexpected change in usage of memory can be detected and analyzed for intrusions. According to Alshamrani, one of the ways malwares can evade detection is by running within the memory of the end system. In doing so, it leaves no trace except for increase in memory usage which can be identified if monitored (Alshamrani et al., 2019).

2. Disk Monitoring: Disk monitoring monitors all malicious activities through firewall, blacklisting, content filtering and anti-virus software. Monitoring the Control Processing Unit (CPU) usage for each end system in the network also helps identify any abnormal behavior that could indicate an attack.

3. Packet Monitoring: This method monitors network traffic by examining streams of data packets flowing between computers on a network or between networked computers to the larger internet, capturing TCP/IP traffic. According to Chu's paper, most packet monitoring methods available on the market use the sniffer method to monitor real-time packet transmissions and compare suspicious packets with others used in previous attacks. (Chu et al., 2019). Using this method, the defense system can alert with an immediate warning when an intrusion is detected. However, there is currently no indication of what devices provide the best rate of APT detection. There are generally three types of APT attack detection using packet monitoring: sandbox, full-flow detection, and abnormal network. All have low accuracy and only focus on single stage of an APT attack; therefore, they typically cannot support the full life cycle of the APT attack.

4. Code monitoring: This method involves a review of code (software developed) using static analysis before it is released. Manual code review can help uncover unknown bugs that attackers could exploit to penetrate the system. Monitoring the code in execution, where any abnormal performance in resources and memory can be observed, can also help identify threats early, before they spread.

5. Logs monitoring: This method can be used to detect unauthorized access attempts and other malicious activity. It can also help troubleshoot network problems and identify potential security vulnerabilities. The information collected by the logs such as system logs with timestamp, CPU usage, memory usage, and application execution are correlated to help systems detect attacks, and can also help with post analysis of any attacks using captured logs as evidence.

2.2.2 Detection methods

For APT detection methods, there are generally two known techniques: pattern matching and anomaly detection.

1. Pattern matching: This technique aims to detect network anomalies in general and often uses database access logs relying on behaviors of a process or an application. Do Xuan indicated that pattern matching requires a cumbersome computing system and a complete training dataset, making it not a viable solution for most applications to detect APT attacks (Do Xuan et al., 2020). Virvilis' paper also echoed the shortcoming of pattern matching when attackers use evasive techniques to encrypt and obfuscate their network traffic to and from the C&C servers and bypass the intrusion detection system. (Virvilis et al., 2013).

2. Anomaly detection: This traditional intrusion detection system is signature-based to identify traffic or application patterns that are assumed to be malicious. Anomaly detection compares activities to a normal baseline and has advantages over the traditional signature-based detection, namely, the ability to detect unknown attacks because the norm has been modeled for any deviation to be detected, and the ability to customize normal activity profiles for system, application and network activities (Omar et al., 2013).

In recent years, ML techniques have been used to improve detection (Maseer et al., 2021). The various approaches for detecting APT attacks above have largely failed because of the multi-stage life cycle of APT (see Figure 2).

2.2.2.1

Machine Learning

ML is a sub-field of artificial intelligence (AI) that "gives you the computational process of automatically inferring and generalizing a learning model from sample data. ML studies algorithms and techniques to automate solutions to complex problems that are difficult to program using conventional programming methods." (Quintero-Bonilla et al., 2020) ML algorithms evaluate the network condition by classifying processed data into normal or abnormal classes (Maseer et al., 2021). There are three well-known machine learning models: supervised learning, unsupervised and semi-supervised learning.

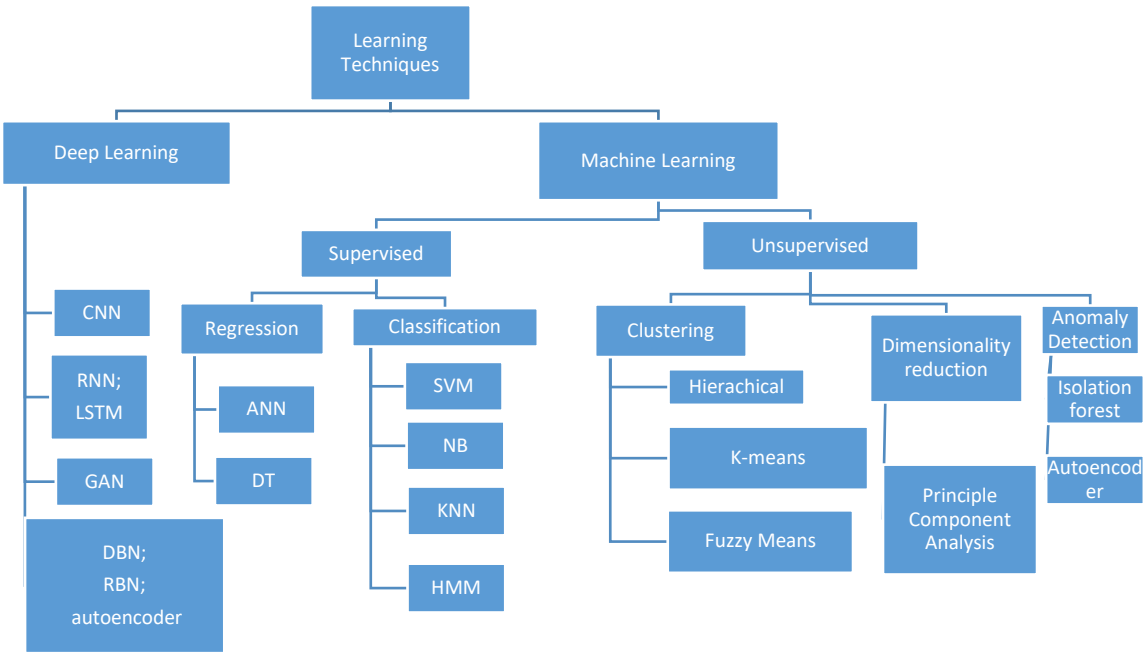


Table 4: Learning Techniques (Quintero-Bonilla et al., 2020; Khaleefa et al., 2022)

Supervised learning

Supervised ML aims to build a model that creates evidence-based predictions. In this model, the dataset used has been tagged/labelled and, by using algorithms, it will be trained or "supervised" to generate analytical predictions (Khaleefa et al., 2022). The

model can be improved over time using clearly labeled dataset as inputs and outputs (known responses to the data). Supervised ML uses both classification and regression techniques to develop predictive models. Classification assigns test data to certain groups (e.g., children vs. adults); regression determines the relationship between the dependent and independent variables. This can be a useful tool for predicting numerical values.

Unsupervised learning

According to IBM, this ML technique analyzes and cluster unlabeled datasets. Without the help of humans, the unsupervised learning algorithms discover hidden patterns or data groupings. The goal of unsupervised learning is to extract valuable information from a large amount of new data, without requiring human intervention to label the dataset. One drawback of this type of ML is that it is computationally intensive, and without human interaction, the unsupervised learning may produce erroneous findings (Khaleefa et al., 2022). *Clustering*: a data mining approach and an unsupervised learning model that uses similarities and differences to classify unlabeled data. It is also typically implemented as a non-supervised learning method, with hierarchical clustering (divisive and agglomerative), K-means (select input data into K clusters), and Fuzzy means (randomly select the number of clusters). *Dimensionality reduction*: This methodology is used when the number of features in a dataset is too large and also an unsupervised learning. Principal component analysis (PCA) is a statistical method used to reduce dimension when there are large number of variables, by, for example, generating a T scoring matrix with correlations between variables that are displayed. *Anomaly Detection*: this methodology identifies rare and abnormal instances in a dataset. Two

most commonly used techniques are Isolation Forest and Autoencoder. This methodology is mostly used for features extraction, social science and medicine.

Semi-supervised learning

Semi-supervised machine learning is a combination of supervised and unsupervised learning. It uses both labeled data and unlabeled data. Usually, smaller portion for unlabeled data and bigger portions of labelled dataset are used due to the need to avoid issues with finding a large number of labeled data only.

Deep learning

DL is a subset of ML that uses multiple layers to extract high-level features from input and is relatively more complex than the other ML algorithms. Lecun (Lecun, 2015) characterizes DL as representation-learning methods with multiple levels, with simple but non-linear modules that each transform the representation at the first level—the raw inputs level—into higher representation. With enough of these transformations, very complex functions can be learned. DL evolved from ANN with deep neural network architectures and improved learning capabilities (Janiesch et al, 2021). Bodstrom et al (Bodstrom et al., 2019) have shown that certain anomalies and behaviors can be detected when combining different types of tailored DL-methods. Their research has also shown that DL methods provide the best performance for APT detection.

2.2.3 Mitigation methods

Mitigation methods are classified into proactive and reactive methods.

1. Proactive methods: These methods are intended to deceive the attacker, or open up the attack surface and increase the challenges to attack.

A) Honeypot: These are dummy systems that are placed alongside the production systems on the network. As a proactive method, a honeypot can also provide

additional security monitoring capability for the blue team (in penetration testing, the blue team is responsible for maintaining internal network defenses against cyber attacks) while diverting an attacker's intended path. John et al. (John et al., 2017) describes an approach where the honeypot is tied to an alarming system, such that early detection or alerting of an intrusion can be deployed. A honeytrap can also be configured to increase the chance of the attacker being identified by using alerts implementing the KFsensor tool based on a Windows architecture and coding a suitable Perl-based alerting module. The simplicity of honeypot makes it less resource-intensive. However, it is limited within the application domain, and according to Khaleefa, honeypot does not provide real-time detection and prevention. (Khaleefa et al., 2022).

B) Moving Target Defense (MTD): This is a proactive method, where the goal is to proactively defend against reconnaissance. Moving constantly between multiple configurations in a cyber-system increases the uncertainty for the attacker, and can be accomplished through measures such as changing network configuration, updating open/close network ports, scanning and patching software. The drawback of MTD is that with sufficient time, hackers can eventually predict the movement and design a successful attack (Sengupta et al., 2020).

2. Reactive methods: These methods are intended to identify possible attack scenarios based on vulnerabilities currently in the system and use analyses—such as graphical analyses—to predict an attacker's possible path.

A) Graph analysis: This type of analysis consists of graph construction and attack graph analysis. Attack graphs are constructed with, among other things, vulnerability

information, time efficiency considerations, cost-effective network hardening, logical dependencies and divide and conquer considerations (Do Xuan et al., 2020). Do Xuan indicated that graph analysis is ineffective for APT detection because APT attacks usually try to emulate behaviors that mimic normal activities. Using the attack graph for anomaly analysis would therefore be difficult.

2.3 APT Datasets

With the increase of ML/DL techniques for intrusion and APT detection, researchers have a need for a common evaluation methodology. Without a common dataset, it is difficult for researchers to benchmark any baselines to use for reviewing algorithms to improve on. Datasets that can be used for evaluation, and APT datasets in particular, are also rare and very restricted, which make training and testing ML/DL algorithms less effective with unrealistic datasets. The reason for this is understandable: none of the businesses and organizations that have experienced APT attacks are eager to share the network captures of the attack. Not only do they want to avoid disclosing the fact that their organization has been attacked, they also do not want their business secrets and sensitive information leaked (intentionally or inadvertently). They also do not want any sensitive technical information to be exploited, such as their network configurations, IDS/IPS implementation or boundary defense solutions, as this information could be exploited by future attacks. The most commonly used datasets have been outdated for over 20 years and most have been derived from the same dataset—the DARPA 1998 datasets.

Three types of datasets have been discussed in Alshamrani and Skopic's papers (Alshamrani et al., 2019; Skopic et al., 2014): realistic data, synthetic data and semi-synthetic data.

Realistic data: Realistic data is the most useful and accurate type of dataset, as it enables testing under realistic conditions and can provide the most realistic outputs and accurate predictive models. However, drawbacks include privacy concerns, the introduction of potential vulnerabilities while running attack simulations on the production system and, because of the ever-changing nature of APT, require occasional updating. In addition, this is typically the most expensive dataset to obtain.

Synthetic data: Synthetic data is artificially created in a controlled environment with the desired network setup. The primary drawback for this type of dataset is the lack of noise that exists in a real network; the dataset becomes too simplified and may lead to unrealistic detection.

Semi-synthetic data: Semi-synthetic data is a combination of realistic data and synthetic data. The drawback to this type of data is similar to that of synthetic data: the simplified dataset will lead to unrealistic detection and can potentially create a biased dataset following an insufficiently accurate and not precise semi-synthetic user model. However, there are significant advantages as well. One is cost, as it is cheaper compared to a realistic dataset. Further, semi-synthetic data is scalable and adaptable to environmental conditions, making it relatively easier to analyze with proper feature selection and attack detection.

Below are some examples of the most popular datasets used with IDS and for IDS with ML. (Stojanovic et al., 2020; Khaleefa et al., 2022; Maseer et al., 2021; Choobdar et al., 2022; Sharafaldin et al., 2018))

DARPA98: Created by Lincoln Laboratory in 1998, this dataset was constructed for network security analysis and consists of a collection of communications between source and destination IP addresses. The dataset includes email, browsing, File Transfer Protocol (FTP), Telnet, Internet Relay Chat (IRC), and Simple Network Management Protocol (SNMP) activities. It also includes various attack data, including DoS, guess password, buffer overflow, remote FTP, sync flood, NMAP and RootKit. The dataset does not include real-world network traffic and it is outdated for effective modern evaluation of any IDS. According to some scholars, another significant drawback is a lack of actual attack data records.

KDD'99: Created by the University of California in 1999, this dataset is an updated version of DARPA98, generated by processing the tcpdump portion. It contains different attacks such as neptune-DoS, pod-DoS, smurf-Dos and buffer overflow. The benign TCP/IP traffic is merged with the attack traffic in this simulated military environment. However, the dataset has a large number of redundant records and leads to skewed testing results.

NSL-KDD: This dataset was created using the KDD'99 dataset. It addresses some of issues in the KDD'99 dataset pointed out by McHugh (McHugh, 2000). However, it still lacks updated versions of attacks and introduces bias to frequency records. The dataset contains 41 features extracted from raw network data from DARPA 98. It contains 4 gigabytes of network traffic recorded over 7 weeks, and can be processed into about 5

million connection records. The NSL-KDD does have sufficient records for training and test sets. The dataset reduces the data size of KDD'99 by deleting duplicate records. Maseer et al (Maseer et al., 2021) indicated that over 65% of ML researches on IDS used KDD'99 and NSL-KDD datasets.

UNSW-NB15: This dataset was created in 2015 in a simulated network environment. The raw network packets were generated using the IXIA PerfectStorm program at the University of New South Wales (UNSW) Canberra's Cyber Range Lab to produce a combination of genuine current regular activities and synthetic contemporary attack behaviors. 100 gigabytes of raw traffic was captured using the tcpdump program over a 31-hour period. The dataset has 175,341 normal classes, and 82,332 anomaly classes and 49 extracted features. Ffuzzers, backdoor, DoS attack, exploits, analysis, reconnaissance, shellcode, worm and generic attacks are included in the dataset. The dataset is also available in both packet and flow-based formats with additional attributes.

NGIDS-DS: Created in 2016 in a simulated network environment, this dataset contains seven features extracted from raw network data and nine features extracted from log files. The dataset maintains both packet-based network communication and host log files. It contains attack families such as backdoor, DoS, exploits, generics, reconnaissance, shellcode and warm.

TRAbID: Created in 2017, this dataset contains 16 distinct IDS assessment scenarios, all of them are collected in a simulated environment with one honeypot server and one hundred clients. Each scenario lasts 30 minutes, for a total of 8 hours. Port scanning and DoS are the two attacks captured.

CICIDS2017/CICIDS2018: This dataset was created by University of New Brunswick. It simulates real-world network data in PCAP files, and uses CICFlowmeter to extract 78 features and 79 labels (CSV). The dataset scenario is 5 days, with abstract characteristics of 25 users according to the HTTP, HTTPs, FTP, SSH and email protocols. It also contains attacks of brute force FTP, brute force SSH, Dos, heartbleed, web infiltration, botnet, and DDOS attacks which are not covered in any of the other previously-mentioned datasets. The 2018 dataset consists of network traffic and log files in a larger emulated network, with the victim organization of 5 departments with 420 machines and 30 servers for 18 days of operation. Both datasets have the same number of features and contain both packet and flow-based entries. The CICIDS2017 dataset has the most distinguished features in terms of realism and reliability for evaluation.

CICIDS2017/CONTAGIO APT dataset: this dataset was created by Dr. Stojanovic and Dr. Hofer-Schmitz at FH Joanneum in Austria. The dataset is semi-synthetic, and merges the CICIDS2017 dataset with CONTAGIO APT datasets (a collection of malware samples, threats, observations and analyses, and contains a total of 36 APT samples). For compatibility, features are extracted from six preselected contagion files and CICIDS2017 PCAP files with the CICFLOWMETER tool. (Nesuschmied et al., 2022)

2.4 APT Detection Literature Review

2.4.1 Detecting APT Attacks with Deep Learning

In Xuan and Huong's research (Xuan & Huong, 2022), they propose building a behavior profile by synthesizing and extracting APT malware behavior based on event log IDs. Events are collected by a system monitoring tool called Sysmon, where the events include registry events, network connection, and File Deletion events. Once the

APT behavior profile is built, the Graph Isomorphism Network (GIN) deep learning Graph Network is used to detect APT malware. The dataset for this research comes from various sources for normal data and APT malware, which were also in different file formats. The GIN model has an accuracy and precision of about 93% with a false positive rate at 13.23%The research fails to provide the time it takes to train and test the GIN model with its small and balanced dataset. As the authors have also pointed out, “there is a trade-off between computation time and detection efficiency (Xuan & Huong, 2022, p 14022)”.

Abdullayeva proposed an autoencoder-based deep learning with softmax regression algorithm to detect APT attacks. The algorithm has a high accuracy of 98.3%, but also a higher false positive rate at 45%. The author used MalwareTrainingSets for training the model, and the data used were split into 3 categories that included APT1, crypto, and other attacks. The limitation of this dataset was that it did not contain any normal data, so the proposed algorithm could not be used in real-time APT detection for any enterprise networks.

2.4.2 Detecting APT Attacks with Supervised Machine Learning Algorithms

Rather than use deep learning algorithms, other researchers reviewed supervised ML algorithms for detecting APT attacks. Ghafir et al., (Ghafir et al., 2018) propose an APT detection system called MLAPT with 3 phases: threat detection, alert correlation and attack prediction. The detection algorithms include KNN, SVM and DT. It is not entirely clear whether or not the dataset came from an active university network feed, which would not be able to validate what are the actual APT attacks. The MLAPT has an accuracy of prediction of APT attacks at 84.8%, with a low false positive rate of 4.5%.

One drawback described by the authors is that the detection modules require a continuous update because they do not work consistently.

Random forest is a different supervised ML algorithm that has been used by Xuan (Xuan, 2021) for APT detection. Xuan used thirty features from both APT domain and APT IP for training random forest. The dataset used are pieced together by random sources (e.g., dataset of Benign collected from various domain names on the Internet, and malicious attacks on collected from CTU Malware Botnet database and Mila database). The RF algorithm obtained an accuracy of 94%, and a lower precision of 80%, with a low false positive rate of 3.72%. The features selected were limited to only domain and IP information—and lack, for example, flow data and https information—and therefore are not sufficient to characterize unique APT features.

2.4.3 Detecting APT Attacks with Unsupervised Machine Learning Algorithms

The limitations of APT detection in the above reviewed literature demonstrates a lack of real-time enterprise network with actual APT attacks. To mimic as close to possible a real-time network with actual APT attacks, Neuschmied et al. (2022) created a semi-synthetic dataset that combined both CICIDS2017 and Contagio APT attack datasets together. The researchers selected nineteen APT features using descriptive analysis, boxplots, histograms, and correlation analysis. The detection algorithms used include autoencoders (AE), variational AE, autoencoder based on convolutional networks (AE-CNN), one-class support vector machine (OCSVM) and combined AE+VAE. The results indicate that AE-CNN has the best performance comparing to the other algorithms. It has lower accuracy of detecting zero-day (anomaly with APTs embedded) at 81.7%, and a low false positive rate of 4.2%, where the training time is around 14

seconds. One gap in this research is that some of the performance metrics are lower, including an F1 score of 35.7% and recall at 25.4%, which is likely caused by the highly unbalanced dataset.

2.5 Summary and Conclusion

This chapter reviewed a contextual background of APT and its detection techniques. The literature review provided a heuristic overview of APT. It covered the characteristics and life cycle of APT and the detection techniques used by various intrusion detection systems, with APT as the special case. Some of the historical APT attacks were also reviewed with respect to the tactics and specific nation states that implement them. The anomaly detection of ML algorithms and techniques were covered in this chapter, as well as the datasets that are used for APT research.

Literature review has been performed on some recent APT detection research. Various ML algorithms including deep learning, supervised and unsupervised are used for attempting to detect APT with better accuracy and precision. Most of the research lacks realistic APT datasets to train and test, and other research does not provide APT unique features that can be used to train the ML models more efficiently. Lastly, some of the performance metrics obtained can be improved further.

Chapter 3—Methodology

3.1 Introduction

This chapter describes the methodology used and implemented in this praxis. There are three steps to this methodology. The first step is to pre-process the datasets for use in the analysis. In the next step, the datasets are processed for feature selection so a training model can be trained with the selected features, which reduces the dimensionality of the input. In the last step, once the training model has been established, testing and validation will take place for evaluation of various ML and DL models. The merged dataset is split as follows: 70% for training, 10% for testing and 20% for validation.

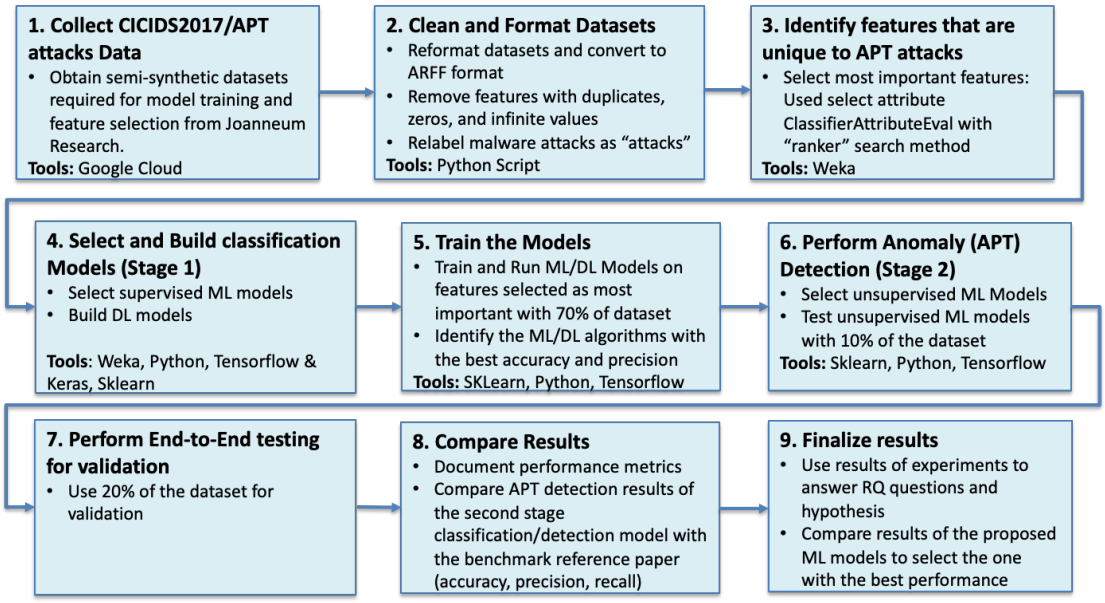


Figure 4: Graphical Model of 2-Stage Detection/Classification Architecture

3.2 Data Collection

This praxis uses a semi-synthetic data developed by an Austrian research institute named Joanneum Research. The datasets were initially created by Dr Hofer-Schmitz and

Dr Stojanovic from the Joanneum Research, and used in the referenced Journal Articles Neuschmied et al., 2022; Stojanović et al., 2020; Hofer-Schmitz et al., 2021.

The data is a combination of the CICIDS2017 dataset and the CONTAGIO dataset (see section 2.3 for background on the CICIDS2017 the CONTAGIO datasets).

The CICIDS2017 dataset simulates real-world network data in PCAP files (network packet capture used to analyze network characteristics). The CICIDS2017 dataset includes various attacks, such as brute force FTP (File Transfer Protocol), brute force SSH (secure shell), DOS (Denial of Service), heartbleed, and web infiltration, among others; however, it does not contain APT attacks.

CICFlowmeter (a network traffic flow generator and analyzer) is used to extract 78 features to CSV (comma separated value) format. The CICflowmeter has been used in prior research efforts directed toward intrusion detection and APT analysis (Maseer, et al., 2021., Khaleefa, et al, 2022, etc), and can be used to read in PCAP files and extract features from TCP/UDP network traffic, including flags, subflow information, inter-arrival and idle-based features. Some PCAP files include source and destination IPs, protocols and length of packet information.

The CONTAGIO dataset contains a collection of Malware samples, threats, observations and analyses, consisting of a total of 36 APT samples, where these APT samples are also in PCAP files which is also extracted to CSV format using the CICFlowmeter.

The researchers provided six datasets in total. The metadata for the three validation and test datasets are summarized in Table 5 and Table 6 respectively. The

datasets are created by inserting CONTAGIO APT network flow data into the CICIDS2017 dataset at different times. The CICIDS2017 captures network traffic from Monday through Friday, July 3 through July 7, 2017, running from 9:00 A.M. to 5:00 P.M. Monday is a normal day and only includes benign traffic. Therefore, in both provided validation and test datasets, there are no "CICMONDAY." Under the "Label" column in Tables 5 and 6, the various attacks are identified for the days that follow: Tuesday includes brute force, FTP-patator and SSH-patator attacks; Wednesday includes DOS, DDOS and Heartbleed port 444 attacks during various morning time slots; Thursday includes brute force, XSS (Cross-site scripting) and SQL injection attacks in the morning and Infiltration and Meta Exploit in the afternoon; and Friday includes Botnet, Portscan and DDOS attacks. In Tables 5 and 6 below, the highlighted rows are APT attacks from the CONTAGIO dataset that were injected into the CICIDS 2017 dataset. The "Number" column indicates how many instances of the attack were captured in the network traffic. As the table indicates, there are many "benign" instances—i.e., normal activities—in the network.

.

Filename: CIC_thu_1150.csv

Label	Number
benign	456756
brute force	1493
infiltration	36
sql injection	21
trojanpage	13
xss	661

Filename: CIC_wed_1240.csv

Label	Number
DoS	252661
benign	440031
heartbleed	11
nettravler	37

Filename: CIC_wed_1241.csv

Label	Number
9002	3
DoS	252661
benign	440031
heartbleed	11

Table 5: Details of the Validation Datasets Provided By
Dr. Hofer-Schmitz and Dr. Stojanovic (APT attacks highlighted)

Filename: CICFri.csv

Label	Number
Lurk	253
benign	414362
bot	1959
ddos	128360
portscan	158602

Filename: CICTue.csv

Label	Number
benign	432072
enfalLurid	14
ftp-patator	7938
ssh-patator	5898

CICTueCookies.csv

Label	Number
benign	432072
ftp-patator	7938
ssh-patator	5898
trojanCookies	1037

Table 6: Details of the Test Datasets Provided By
Dr. Hofer-Schmitz and Dr. Stojanovic (APT attacks highlighted)

3.3 Clean and Format Dataset

The datasets provided by Joanneum Research went through three steps:

1) Reformatted the data for upload to WEKA

The datasets contained certain formatting issues which prevented WEKA—a data mining software provided by the ML group at the University of Waikato—from reading them. A python script was written to reformat the datasets (see Appendix A – Figure 1)

2) The datasets contained number of features with zero or infinity values. These columns were removed from the file because they do not contribute any values for ML model training or feature selections. The columns are detailed below:

> Bwd PSH Flags; Bwd URG Flags; CWR Flag count; ECE Flag count;
> FWD Bulk Rate Avg; Fwd Bytes/Bulk Avg; Fwd Packet / Bulk Avg;
> FWD URG Flags; URG Flag.

3) Various malware attack labels excluding APT were relabeled to just "attack" and various APT attacks were just labeled to "APT" in order to better distinguish APT attacks for training the models.

3.4 Feature Selection – Extract Features

Feature selection is an important step for building the training model. Some researchers have used Minitab to identify correlated features (Hofer-schmitz, et al., 2021), while others have used normalization to "bring all attribute values to the same scale" in order to determine significant features (Maseer, et al., 2021). In the feature selection applied herein, Minitab was initially used to perform a correlation analysis between all 69 features, although no consistent results were obtained. However, WEKA includes the feature selection function "select attribute: ClassifierAttributeEval" with "Ranker" search method using entropy, which was used in this praxis.

"Ranker" ranks all of the features from most important to least important. The top ten significant features to distinguish APT attacks were selected for use in training the algorithms (see Appendix A – Figure 2). The Ranker function was then run on the three validation datasets, and the analysis returned the same ten highlighted features below in

Table 7. In addition, the three datasets were merged, and the same WEKA function was run again on the merged datasets, resulting in the same ten features being outputted as significant to detecting APT attacks. To further validate the features selected in Table 7, Du et al.'s paper was reviewed, which describes strong features of APT that are screened by comparing the change of the entropy of the attack source after removing a certain feature (Du, et al, 2018). Features generated from both the WEKA function "Ranker" and Du et al.'s selection were very similar, and thus served to confirm the selected features are the best and most representative for use in this Praxis. The tenth feature (flow bytes/second) was withdrawn because it contained an infinity value that WEKA cannot process (This was not flagged was part of data clean up).

Attribute	Type	Description	If used for future training
Total Length of Fwd Packet	integer	Total size of packet in forward direction	Y
Dst IP	string/integer	IP address for packets destination	Y
FIN Flag Count	integer	Number of packets for FIN	Y
Dst Port	integer	Destination port	Y
Down/Up Ratio	real	Download and upload ratio	Y
Bwd Packet Length Std	real	Standard deviation size of packet in backward direction	Y
Bwd Segment Size Avg	real	Average size observed in the backward direction	Y
Bwd Packets/s	real	Number of backward packets per second	Y
FWD Init Win Bytes	integer	The total number of bytes sent in initial window in the forward direction	Y
Flow Bytes/s		Number of flow bytes per second	N(INF as value)

Table 7: Feature Selection Table

When building the training model with the nine features, it was discovered that the Dst IP (destination IP address), once converted to decimal value from string, reduced the training model size significantly, e.g.: 398KB reduced to 2.3KB. Such a significant

drop can help improve performance efficiency and reduce processing time (See Appendix A, Figure 3).

3.5 Select and Build ML Models

Three seperate approaches of detecting APT attacks were tried for selecting a solution. Each of these approaches l are detailed below:

3.5.1 Approach1: A Single Stage ML/DL model

The objective of approach 1 was to detect APT attacks using three label classifications with a single stage of ML/DL model: benign, APT and attacks. All datasets in the validation folders CIC_thur, CIC_wed_1240 and CIC_wed_1241 (Table 5) were used for training the models, and the CIC_tue dataset in the test folder (Table 6) was used for testing. WEKA 3.8 was installed for MAC OS using an Intel Processor.[1]

The ML/DL models used for approach 1include NaïveBayes (NB), Decision Tree J48, K Nearest Neighbors (KNN), Support Vector Machine (SVM) and Neural Network (NN); and DL algorithms Convolution Neural Network (CNN) and Recurrent Neural Network (RNN). Details concerning the configuration and setup of the algorithms, and the subsequent results, are provided in Appendix B.

For the first approach, with three classification labels, given the highly imbalanced datasets, the best-performing algorithms for detecting APT were both NB and DT supervised ML algorithms (NB has an accuracy of 87.5% and a recall of 64.3%, and DT has an accuracy of 99.9% with a recall of 14.3% in APT detection). The fastest-performing algorithm for combined training and testing time was NB. Many algorithms did not provide conclusive results sufficient to indicate whether they could improve APT

[1] Note that there are different software packages for MAC OS with an ARM processor.

detection. Some demonstrated high accuracy but lacked adequate precision and recall. Imbalanced and/or skewed datasets have been identified as deficiencies in several papers for ML algorithms, where the predicted probability has been influenced when there is less data, thereby creating a biased prediction in favor of the larger class. Normalizing the datasets may be one possible solution to correct unbalanced datasets (Frank et al., 2006). Yijing et al. have proposed another solution using multiple classifiers to reduce the imbalanced datasets (Yijing et al, 2015). Approach 2 was inspired by Yijing's paper using multi-stage multiple ML/DL algorithms to reduce the imbalanced datasets and improve APT detection results. The results from first approach (using WEKA for ML algorithms and TF for DL algorithms for APT detection) indicated high accuracy for only two ML algorithms: NB and DT. Most of the other algorithms had inconclusive precision, recall, and other metrics due to unbalanced datasets, which was primarily caused by a high degree of skew that contained higher numbers of "benign" instances when compared to numbers of APT instances.

3.5.2 Approach 2: Using a Two-Stage Supervised ML Models

The second approach was conducted to determine if the highly skewed datasets could be addressed. The experiment focused on using a two-stage APT detection architecture with supervised ML algorithms.

The first stage leveraged supervised ML algorithms to detect and differentiate the malicious attacks from the benign ones. The second stage was performed to further detect APT attacks and further distinguish such attacks from other non-APT attacks (such as, e.g., DDOS and SQLinjection) by using supervised ML algorithms for both stages. The first stage demonstrated high accuracy and precision (above 95%); but the second stage

yielded inaccurate results due to unbalanced datasets, causing uncertainty in the performance metrics.

For the first stage, KNN, Gaussian NB and Decision Tree were used to detect and differentiate malicious attacks from benign network activities. The Figure 5 shows accuracy, precision, false positive rate (FPR) and recall for the three algorithms after the first detection stage.

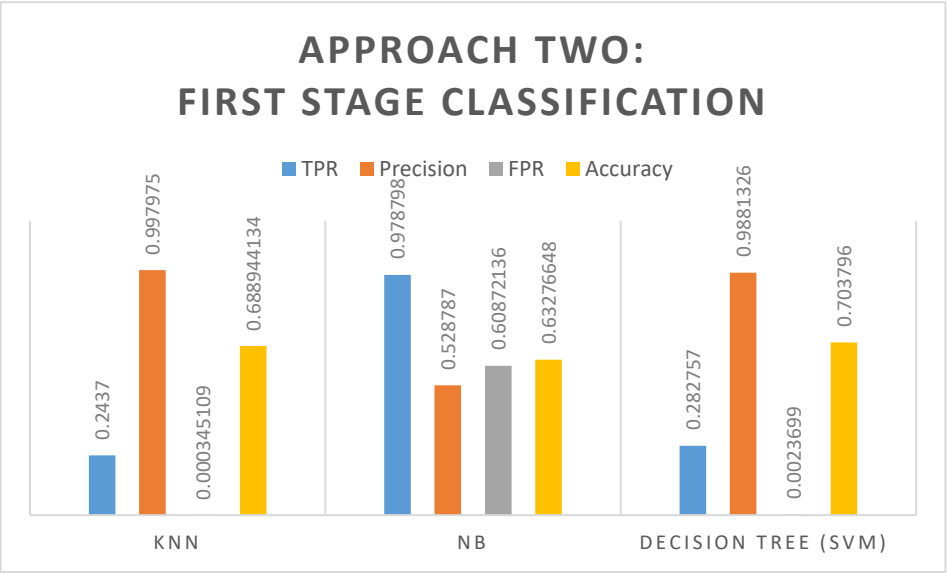


Figure 5 APT Detection for Stage 1 of the Anomaly Detection

The results in Figure 5 demonstrate that both KNN and Decision Tree performed well with higher precision and accuracy, and with very low false positive rate. Decision Tree, however, had the shortest training and testing time compared to KNN (Table 8). Therefore, for the second stage, both KNN and Decision Tree were selected to re-run for detecting APT against other non-APT attacks.

	Training time (secs)	Test Time (secs)
KNN	3.31897	164.0317
NB	0.8714	0.4751
Decision Tree	1.4394	0.5123

Table 8: Training and Testing Time for KNN, NB and DT

For the second stage, KNN and Decision Tree were used to distinguish between APT attacks and non-APT attacks. The results of the testing were inconclusive from the KNN-APT combination (see Figure 6). Though accuracy seems high, but the precision is close to zero.

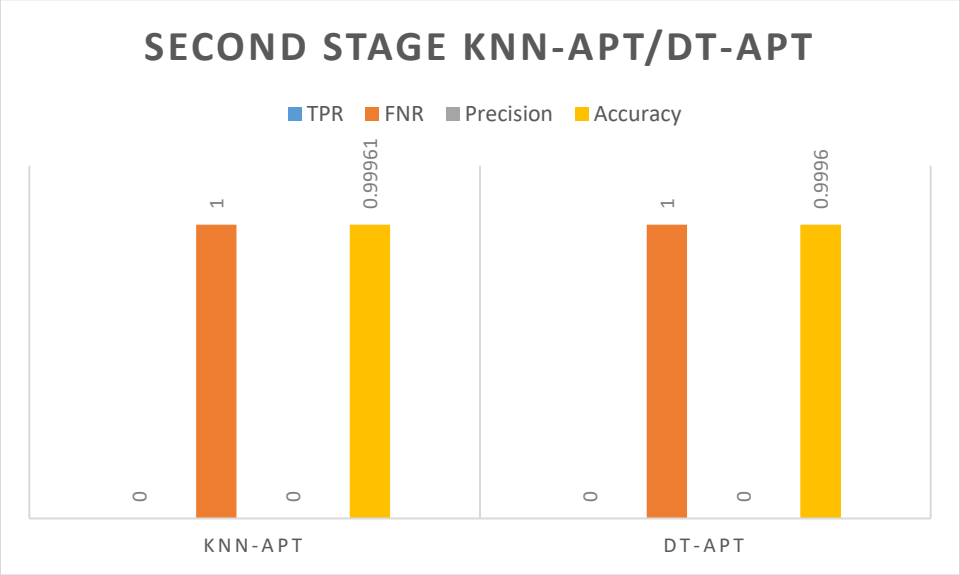


Figure 6: Second Stage KNN-APT and DT-APT Detection

For the second approach, two-stage supervised ML algorithms did not provide sufficient levels of precision and accuracy of detecting specific APT attacks, and did not resolve the overfitting issue.

3.5.3 Approach 3: A Two-Stage Supervised & Unsupervised ML Model

Approach 3 used unsupervised ML models in a the two-stage APT Detection Architecture for the second stage, rather than supervised ML algorithms from the second approach. Unsupervised ML was used in an attempt to resolve the low APT detection rate. In this third approach, three unsupervised learning algorithms were used: Isolation Forest, Bayesian Gaussian Mixture and OCSVM. However, OCSVM failed due to high-dimension attributes. Isolation Forest (IF) was selected because it was faster in both

training and testing time when compared to most other ML algorithms, and could perform outlier (anomaly) detection in high-dimensional datasets more efficiently. IF uses binary trees to detect anomalies/outliers. The algorithm recursively generates partitions by randomly selecting a split value in the data. The number of partitions is used to calculate the anomaly score, which is then used to build the isolation tree. For the third approach, the large dataset shown in Table 9 was used for training, testing and validation. This merged dataset was then split as follows: 70% training dataset, 10% testing dataset, and 20% validation dataset. SKLEARN is a machine learning platform for Python, and was used for the final round of experiments so that built-in ML functions could be utilized. Approach 3 was ultimately used as the main methodology of this praxis.

3.6 Train, Test and Validation

3.6.1 Train

Using the correct datasets to train the ML algorithms can be critical to ensuring accurate APT detection. The first approach reviewed various ML/DL algorithms to detect APT using WEKA and PYTHON; the second approach used two-stage classification/detection models with supervised ML algorithms implementing SKLEARN (a ML modeling tool and library); and the third approach, both supervised and unsupervised ML models are used for the two-stage classification/detection models implementing SKLEARN.

The first approach used training and testing datasets from Table 5 and 6. During the initial experiment, the three validation datasets (Table 5) were merged to create a training dataset for WEKA to train the ML models. The merged training dataset was also intended to be used for validation with either k-cross validation (setting k to 10) and/or the percentage split of training/testing set at 80%/20%. WEKA provides four options

under “test options”: (1) use the training set; (2) supply a test set, which enables the user to choose a testing dataset; (3) cross-validation, where the numeric value of k can be determined; and (4) percentage spilt, where the training/test/validation split can be set. However, it was observed that after running the two models (cross validation and training/testing split), the time required to generate any ML model typically took a significant amount of time (e.g., over 24 hours to run the algorithm, such as the KNN classifier, depending on available computation tools). In order to make WEKA run more efficiently, the netlibNative package was applied to WEKA to accelerate CPU. The supplied test dataset CICtue.csv shown in Table 6 was then chosen to test the trained models. With CPU acceleration, most if not all classifiers can run significantly faster than without using such acceleration.

For the second and final approach, the input CSV files from Joanneum Research were merged to form a single CSV file to serve as the input to the training of ML models. Five of the six datasets, excluding CICTUECOOKIES.csv[2], were combined to form this single dataset for the two-stage Classification/Detection model in preparation for training, testing and validating the various ML/DL models. The table below (Table 9) shows details of the combined large dataset from the following five datasets: Cictue.csv, Cicfri.csv, Cic_wed_1240.csv, Cic_wed_1241.csv, and Cic_thur_1150.csv.

Label	Number of Total Records	Stage 1	Stage 2
Benign	2183252	Benign	Exclude
Brute Force	1493	Attacks	unlabel
Infiltration	36	Attacks	unlabel
SQL injection	21	Attacks	unlabel

[2] The purpose was to avoid redundancy in the Tuesday datasets between CICTues and CICTueCookies datasets.

Label	Number of Total Records	Stage 1	Stage 2
XSS	661	Attacks	unlabel
DoS	505322	Attacks	unlabel
Heart Bleed	22	Attacks	unlabel
Bot	1959	Attacks	unlabel
DDoS	128360	Attacks	unlabel
Portscan	158602	Attacks	unlabel
FTP-patator	7938	Attacks	unlabel
SSH Patator	5898	Attacks	unlabel
Lurk (APT)	253	Attacks	unlabel
EnfalLurid (APT)	14	Attacks	unlabel
TrojanPage (APT)	13	Attacks	unlabel
Nettravler (APT)	37	Attacks	unlabel
9002 (APT)	3	Attacks	unlabel
Total Records	**2993884**		

Table 9: Fully Cleaned Up Dataset

3.6.2 Testing and Validation

For the testing and validation phase, the praxis first evaluated five supervised ML algorithms in the first approach (Naïve Bayes, Decision Tree, SupportVectorMachine (SVM), K-Nearest Neighbors and Neural Networks using WEKA), and continued evaluation with two additional DL algorithms (Convolution Neural Network (CNN) and Recurrent Neural Network (RNN)) using Python3, Tensorflow GPU 2.9.1 and Keras. All of these algorithms were run on the same test dataset, where all metrics were collected for evaluation. For SVM, regardless of the kernel[3] that is being used, the Social Media Optimization (SMO) function cannot complete training. For Nu-SVM, similar errors indicate Nu-Value is infeasible; therefore, SVM was dropped from the evaluation for APT detection.

[3] Kernel is used due to a set of mathematical functions used in SVM providing the window to manipulate data.

Evaluation of the above supervised ML algorithms in the first approach did not achieve over 90% accuracy and precision in APT attacks detection. The plausible reason could be the imbalanced large dataset that has millions records in benign network traffic and only hundreds of records in APT attacks. Both Rennie et al. and Frank & Bouckaert demonstrated that when skewed class sizes are used for training, if one class significantly outnumbers the other class, decision boundary weights become biased. In other words, predictions and outputs will also be biased and lean toward the larger class. (Grank & Bouckaert, 2006; Rennie et al, 2003)

A two-stage APT detection architecture has been designed to address the imbalanced dataset issue and overfitting of the datasets caused by imbalanced datasets. The first part of the detection process distinguishes between benign activity and overall attacks, as well as classifying all anomaly attacks. After classifying all attacks, the second part of the detection process detects APT attacks from regular malware attacks. Table 9[4] shows an estimated 3 million network traffic records, with about 2.2 million being benign, normal traffic records. The dataset is then split 70% for training, 10% for testing for the ML/DL algorithms reviewed during the initial experiment, and 20% for validation of the two-stage classification/detection model.

[4] Table 5 and Table 6 validation and test datasets merged into one dataset excluding CICTUECOOKIES.

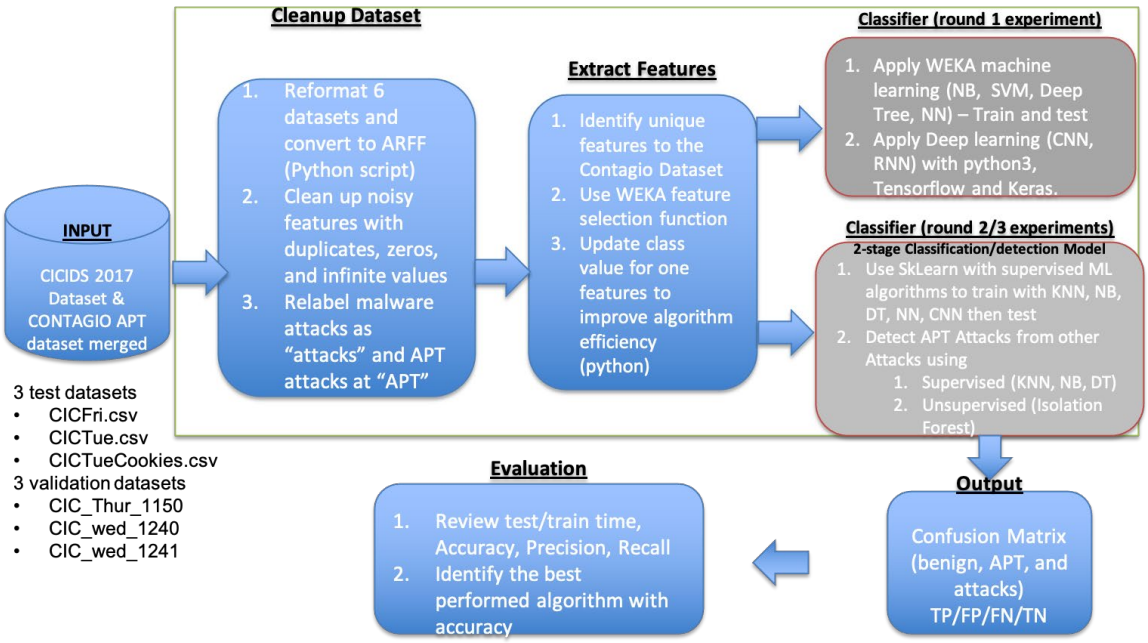


Figure 7A: wo-Stage Classification/Detection Model (see Tables 5 and 6 for Test and Validation Datasets)

The first part of classification was tested with both supervised ML and DL algorithms. Conducting supervised ML for the first stage of the architecture resulted in high accuracy and precision for anomaly detection. For the second stage, both supervised and unsupervised ML algorithms were tested for detecting APT attacks. Here, the unsupervised ML algorithm outperformed supervised ML and was selected for the second stage.

3.6.3 Evaluation

The praxis evaluates quantitative metrics generated from each ML algorithm to determine which ML/DL algorithm can better detect APT attacks, and which yields the most efficient and highest performance. It is critical to use the same set of training, testing and validation datasets in the analyses to ensure a common benchmark for comparison. The praxis will evaluate Accuracy, Precision, Sensitivity, F1-Score, duration

of building the model and duration of testing the model, as defined below (Berman et al., 2019):

Accuracy: Accuracy is the ratio of the correct predictions of True Positive (TP) and the True Negatives (TN) of attacks against the total number of test cases:

Accuracy = TP + TN / (TP + TN + FN + FP)

Precision: Precision measures the accuracy of positive predictions (FP = False Positive) :

Precision = TP / (TP + FP)

Recall/Sensitivity: Sensitivity measures the ability for the algorithm to determine the TP correctly:

Recall/Sensitivity = TP/ (TP + FN)

F1-Score: F1-Score measures the ML model's accuracy. It combines the precision and recall scores of a model:

F1-score = 2* Precision * Recall / (Precision + Recall)

Time to build a model: Time taken for ML/DL to build the model with the specific algorithm – training time.

Testing time: Time taken to complete test the dataset with the specific ML algorithm.

Class	Predicted Positive	Predicted Negative
Class of APT attack (positive)	Predicted Attack as attacks (TP)	Predicted Attack as Normal/ Benign (FN)
Class of Normal (negative)	Predicted Normal/Benign as Attacks (FP)	Predicted Normal/Benign as Normal (TN)

Table 10: Binary Confusion Matrix

In the above table (Table 10), TP means that the model was able to accurately classify the APT attack, and conversely, FP means falsely classifying a benign activity as an attack. TN means the classifier accurately classified normal activities, while FN means the APT attack has been misclassified as normal (Berman et al., 2019).

Chapter 4—Results

4.1 Introduction

This section discusses the results and evaluation of the final approach in the methodology that was selected for APT detection for the praxis. In Chapter 3, three approaches were discussed: (1) first approach using single stage ML algorithms using WEKA for ML algorithms and Python3, Tensorflow (TF) and Keras for two DL algorithms (namely, (CNN) and RNN)); (2) second approach using a two-stage APT detection architecture with supervised learning; and (3) final approach using the same two-stage architecture as the second approach with a combination of supervised ML and unsupervised algorithms that resulted in higher accuracy for detecting APT attacks. Python3, TF, Keras and SKLEARN were also used for the latter rounds of ML training, testing and validation.

4.2 Final approach: Using a Two-Stage Supervised and Unsupervised ML Models

The final approach used unsupervised ML models in the two-stage APT Detection Architecture (Figure 7B) for the second stage, rather than supervised ML algorithms in the second approach. Unliked supervised ML models, unsupervised ML models use unlabeled data to find hidden patterns without the need of human intervention or instructions. The merged large dataset in Table 9 was used for training, test and validation.

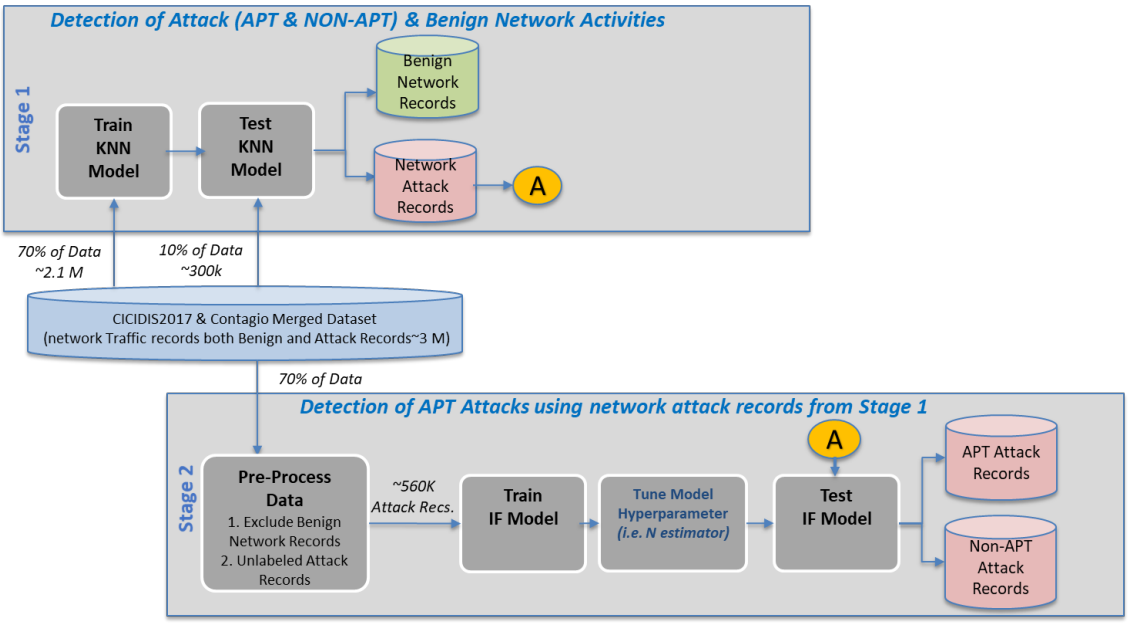


Figure 7B: Two Stage Classification/Detection Model (with Unsupervised ML Algorithm)

First, the merged datasets removed the "label" column to train the unsupervised ML algorithm. For the Isolation Forest, to determine which N_estimator had the best performance for hyperparameter tuning,[5] N = 5, 10, 15 and 20 were tested, and N = 15 generated the best performance results. Figure 8 below illustrates the various performance results obtained. When N_estimator was set to 15 trees, the precision was optimal with a training time of 1.2717 seconds, and the lowest false positive rate. Therefore, the N=15 was selected for Isolation Forest for the remainder of the experiments. Hyperparameter tuning allowed for tweaking the model's performance for optimal results.

[5] "Hyperparameter tuning" refers to the number of base estimators needed to detect anomaly.

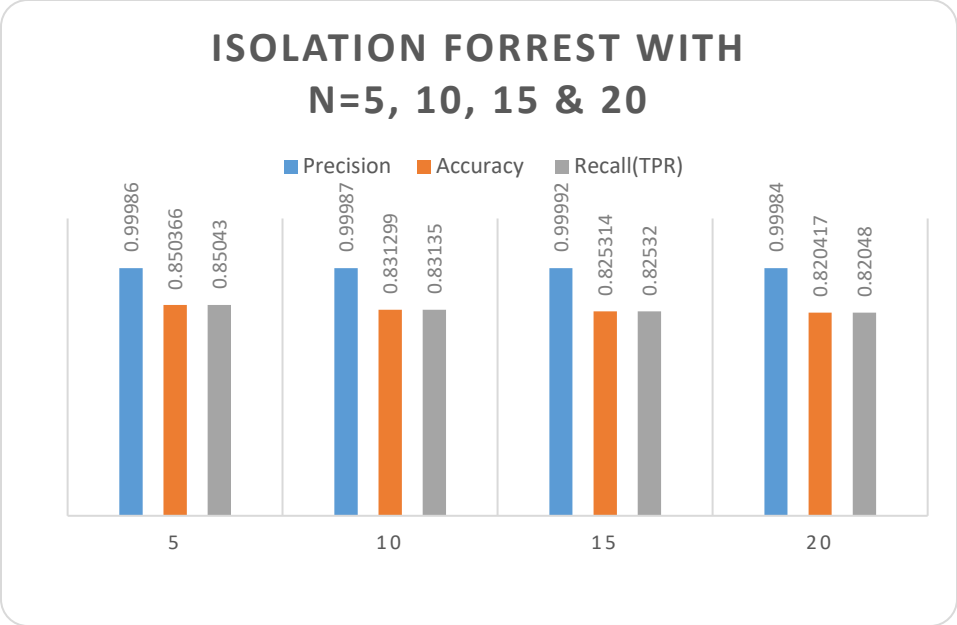


Figure 8: Various N Estimator Performance for Isolation Forest

Using Bayesian Gaussian Mixture, various N components were also tested, which ranged from 1 to 2, as well as random states 1, 10, and 20 (which had no impact on the prediction accuracy). As shown in Figure 9 below, when N=2 (the number of mixture components), the outputs indicated a high level of accuracy and recall overall, with a training time of 79.5 seconds.

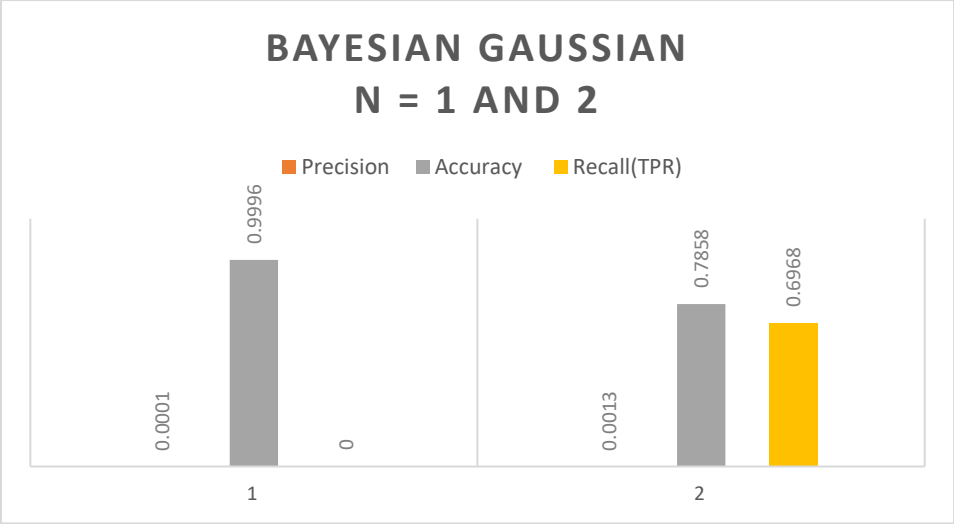


Figure 9: Various N Estimator Performance for Bayesian Gaussian

Out of the 3 unsupervised ML algorithms, Isolation Forest had the highest APT attack detection performance and was selected to be used for the second stage.

The two-stage Classification/Detection model evaluated four ML algorithms for the first stage, with the selected Isolation Forest for the second stage (KNN, Decision Tree, CNN and NN) to distinguish between benign and malicious attacks. As illustrated in Figure 10, both KNN and DT performed with a high level of accuracy and recall. Therefore, both KNN and DT were selected for use in the first stage.

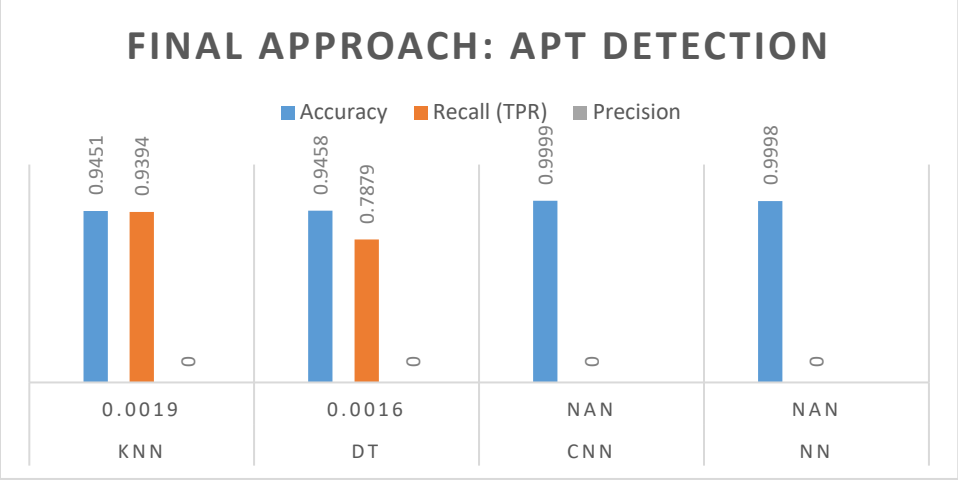


Figure 10: Performance Metrics for APT detection

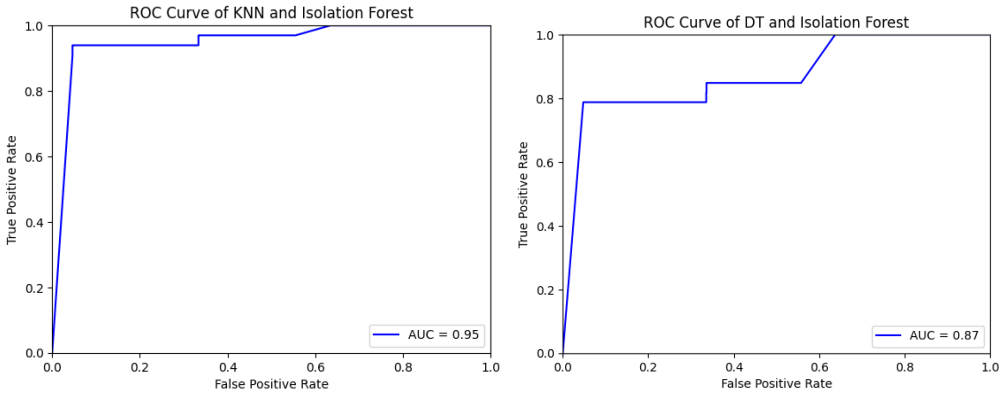


Figure 11: ROC Curves for KNN-Isolation Forect and DT-Isolation Forest

4.2.1 Final Approach: Selection of KNN (First Stage) and Isolation Forest (Second Stage)

The KNN and IF combination outperformed the DT and IF combination in accuracy, recall and training time. The training time for KNN and IF was fifty-seven seconds faster than the DT and IF combination. Therefore, the results indicate that the best detection algorithm available for APT detection is the two-stage APT detection model with both KNN (a supervised ML algorithm) and Isolation Forest (an unsupervised ML algorithm). The accuracy of the surface below the Receiving Operating Characteristic (ROC) diagram (Area Under the Curve or "AUC" (see Figure 11) for the KNN-IF and DT-IF is 95% and 87% respectively, thereby confirming that the KNN-IF combination performs the best for detecting APT attacks.

KNN and Isolation Forest combined had the shortest training time when compared to any other ML model combination, at nine seconds (see Figure 12).

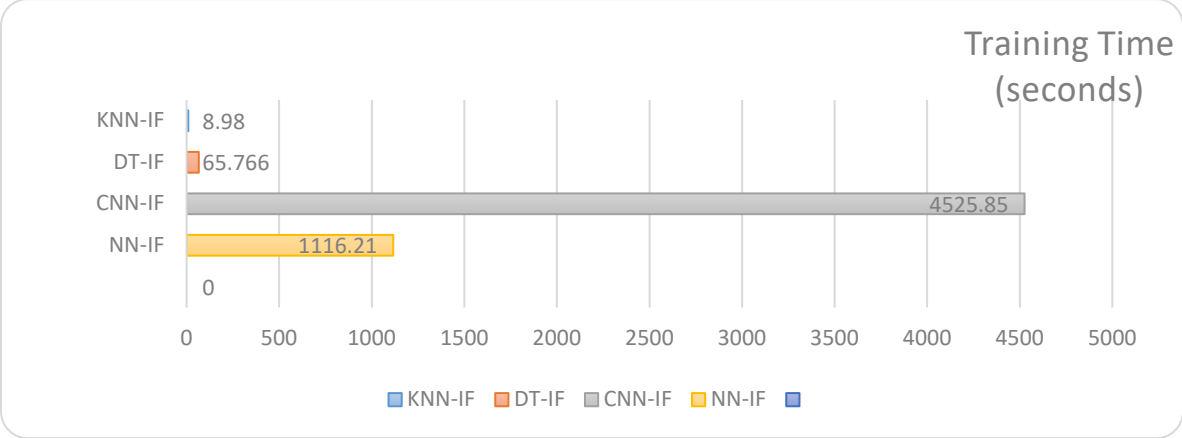


Figure 12: Training time for KNN, DT, CNN, and NN for Round 3 Experiments

4.2.2 Final Approach: Results Summary

The combination of KNN and Isolation Forest (KNN-IF) delivered more than 96% accuracy and 85% recall in the test dataset, and a 96% accuracy and a 61% recall for the validation dataset (see Tables 11 and 12). Both the test and validation results returned

a false positive rate around 3%, which indicates that the probability of misidentifying true positive is very low. This indicates that the model yielded the highest performance, and did so with the shortest training time in both the testing dataset (~8 seconds) and the validation dataset (~9 seconds). Thus, the combined two-stage approach is able to detect APT attacks despite an imbalanced dataset. The red box in Tables 11 and 12 highlights the performance metrics for both 10% testing and 20% validation.

Stage 1	10% Testing					
	Accuracy	Precision	Recall	Training Time (secs)	Predicting time (secs)	
KNN	0.9928	0.9817	0.9919	6.92	59.9	
	False Positive	False Negative	True Positive	True Negative	False Positive Rate	F1 Score
KNN	1477	641	79294	214983	0.0068234	0.986773642
Stage 2						
	Accuracy	Precision	Recall	Training Time (secs)	Predicting time (secs)	
ATTACKS	0.9599	0.9902	0.797			
APT (WEIGHTED)	0.9572	0.9951	0.8571	1.0268	2.1621	
	False Positive (FP	alse Negative (FN	True Positive (TP)	True Negative (TN)	False Positive Rate (FPR)	F1 Score
ATTACKS	678	11183	68724	215810	0.00313	0.883157341
APT	11354	13	15	285013	0.0383	0.920959086

Table 11: 2-Stage KNN-IF APT Detection Performance Metrics for the 10% Testing Dataset

Stage 1	20% Validation					
	Accuracy	Precision	Recall	Training Time (secs)	Predicting time (secs)	
KNN	0.9929	0.9822	0.9916	6.76	116.88	
	False Positive	False Negative	True Positive	True Negative	False Positive Rate	F1 Score
KNN	2933	1358	161589	435891	0.0066838	0.986877617
Stage 2						
	Accuracy	Precision	Recall	Training Time (secs)	Predicting time (secs)	
ATTACKS	0.9431	0.987	0.8002			
APT (WEIGHTED)	0.9642	0.994	0.612	2.2241	4.7733	
	False Positive	False Negative	True Positive	True Negative	False Positive Rate	F1 Score
ATTACKS	1944	18591	144289	436947	0.0038802	0.883837735
APT	18248	26	41	583456	0.0303	0.757569116

Table 12: 2-Stage KNN-IF APT Detection Performance Metrics for the 20% Validation Dataset

4.3 Improvement in APT Detection Model

This praxis benchmarked the performance metrics of detection of malicious attack against the article from Neuschmied et all (Neuschmied et al., 2022) which also used the same input data to show improvement in accuracy, precision and false positive rates, as well as training time. The referenced article used multi-stage autoencoder for APT attack

detection with metrics on anomaly detection and zero-day detection without classifying specific APT attacks.

The table below - Table 13 —derived from the referenced paper (Neuschmied et al., 2022)—shows the following metrics as results for the anomalies detection (the grayed-out AEC is not used for metrics comparison, and the highlighted AE-CNN has the best accuracy and precision):

	Accuracy	Precision	Recall	Training time (secs)
AE	0.872	0.609	0.867	9.56
AEC (autoencoder – custom – built-in labels) *	0.968	0.908	0.919	4.56
AE-CNN	0.905	0.817	0.625	14.17
VAE	0.871	0.635	0.709	243.04
VAE-prob	0.824	0.515	0.884	286.31
AE-VAE-prob	0.869	0.6	0.88	93.25

Table 13: Results of Anomalies Detection (Table 6 of Neuschmied et al, 2022)

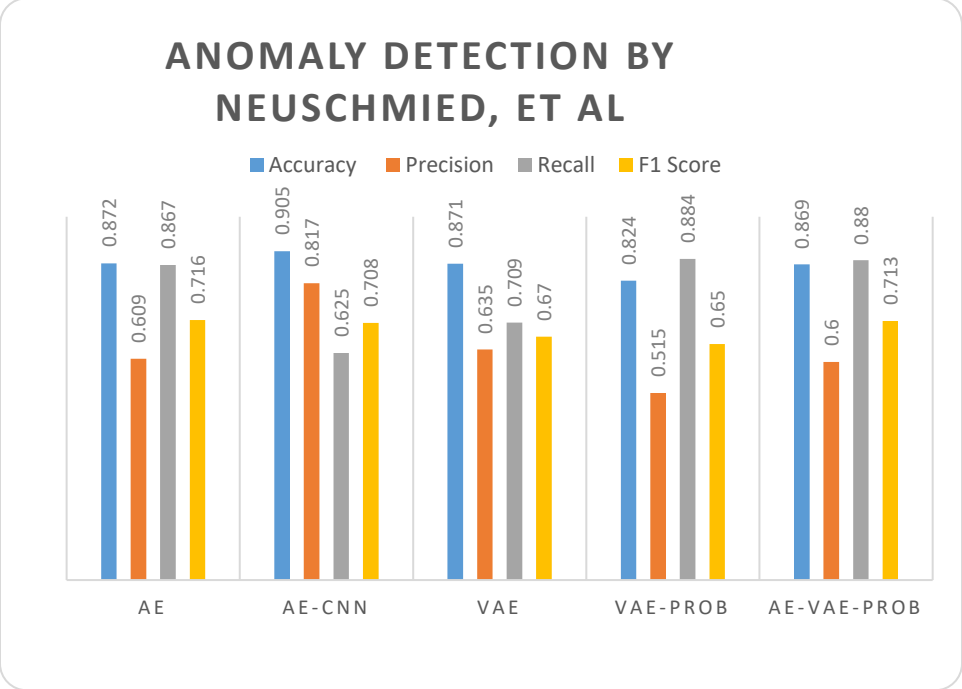


Figure 13: Anomaly Detection Performance Metrics by Neuschmied et al, 2022

For detecting malicious attacks in the referenced paper, a customized AEC[6] was used for testing out the complexity of datasets only (Grayed out in Tables 13 and 14), where the paper has indicated that "AEC stands out as the theoretically best possible result, unfortunately, this is only the case for unrealistic and practically irrelevant scenarios" (Neuschmied, et al., 2022 – page 11). Thus, removing AEC as a true benchmark for comparison, the next-best performing AE-CNN is at 90.5%, which has been highlighted in both Tables 13 and 14.

The below table – Table 14 shows results for the zero-day detection that included APT attacks (TrojanCookies, Eufal_Lurid and BIN_LURK, as well as other malicious attacks FTP patator, SSH patator, Bot, DDoS, and PortScan) from the benchmark paper.

	Accuracy	Precision	Recall	Training time (Secs)
AE (autoencoder)	0.776	0.415	0.297	10.28
AEC (built-in labels)	0.827	0.674	0.256	4.56
AE-CNN (autoeoncoder with CNN)	0.817	0.597	0.254	14.13
VAE (variable Autoencoder)	0.774	0.403	0.28	288.29
VAE-prob	0.742	0.371	0.42	250.7
AE-VAE-prob	0.797	0.49	0.419	79.1

Table 14: Results of Zero Day Detection (Table 8 of Neuschmied et al, 2022)

[6] AEC is the custom Autoencoder that has been error corrected with added labels for an unsupervised algorithm.

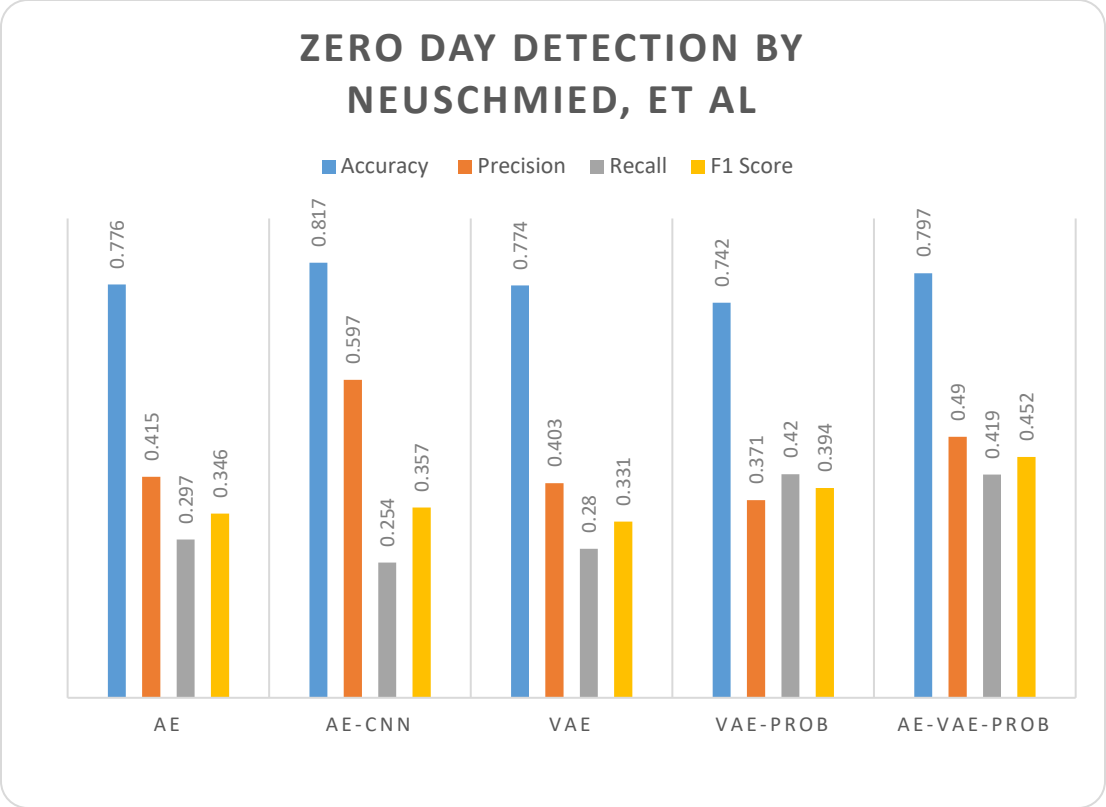


Figure 14: Zero Day Performance Metrics by Neuschmied et al, 2022

Comparing the performance metrics of the referenced paper illustrated in Figures 13 and 14 to the APT detection results illustrated in Figures 15 and 16 from the combined two-stage detection/classification model, this praxis has demonstrated an improvement in both anomalies detection and zero-day detection with respect to accuracy, precision, FPR, recall and F1 Scores (see also Tables 15 and 16). The red box indicates the best performing algorithm for the benchmark paper, and the blue box indicates the praxis performance. KNN-IF in comparison to the Benchmark paper, has higher accuracy, precision and F1-score and the lowest FPR (false positive rate). For the zero-day detection, which has a better presentation of APT attacks, the praxis algorithm outperformed by more than 16% in accuracy, 50% in recall and five seconds faster in overall training time (see Table 16). The praxis continued to differentiate APT attacks

from malicious attacks with high accuracy levels, which has not been accomplished in the previous reference paper.

Anomaly Detection						
	Accuracy	**Precision**	**Recall**	**FPR**	**F1 Score**	**Training time (secs)**
AE	0.872	0.609	0.867	0.12634979	0.716	9.56
AE-CNN	0.905	0.817	0.625	0.03193	0.708	14.17
VAE	0.871	0.635	0.709	0.09276	0.67	243.04
VAE-prob	0.824	0.515	0.884	0.18958	0.65	286.31
AE-VAE-prob	0.869	0.6	0.88	0.13346	0.713	93.25
KNN/IF	.9642	0.9941	0.612	0.0308	0.7576	8.82

Table 15: Improvement of Performance Metrics over Benchmark Paper on Anomaly Detection

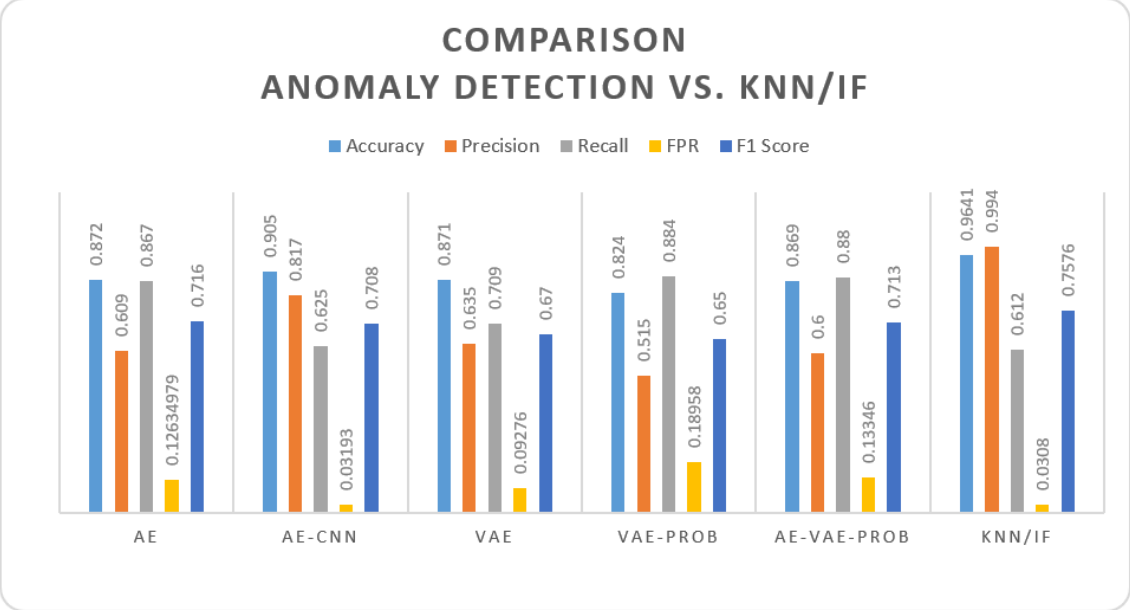


Figure 15: Performance Metrics Comparison of KNN/IF vs. Benchmark Paper (Anomaly Detection)

Zero Day Detection						
	Accuracy	**Precision**	**Recall**	**FPR**	**F1 Score**	**Training time (secs)**
AE	0.776	0.415	0.297	0.10422	0.346	10.28
AE-CNN	0.817	0.597	0.254	0.04272	0.357	14.13

VAE	0.774	0.403	0.28	0.1036	0.331	288.29
VAE-prob	0.742	0.371	0.42	0.17723	0.394	250.7
AE-VAE-prob	0.797	0.49	0.419	0.10871	0.452	79.1
KNN/IF	0.9641	0.994	0.612	0.0308	0.7576	8.82

Table 16: Improvement of Performance Metrics over Benchmark Paper on Zero Day Detection

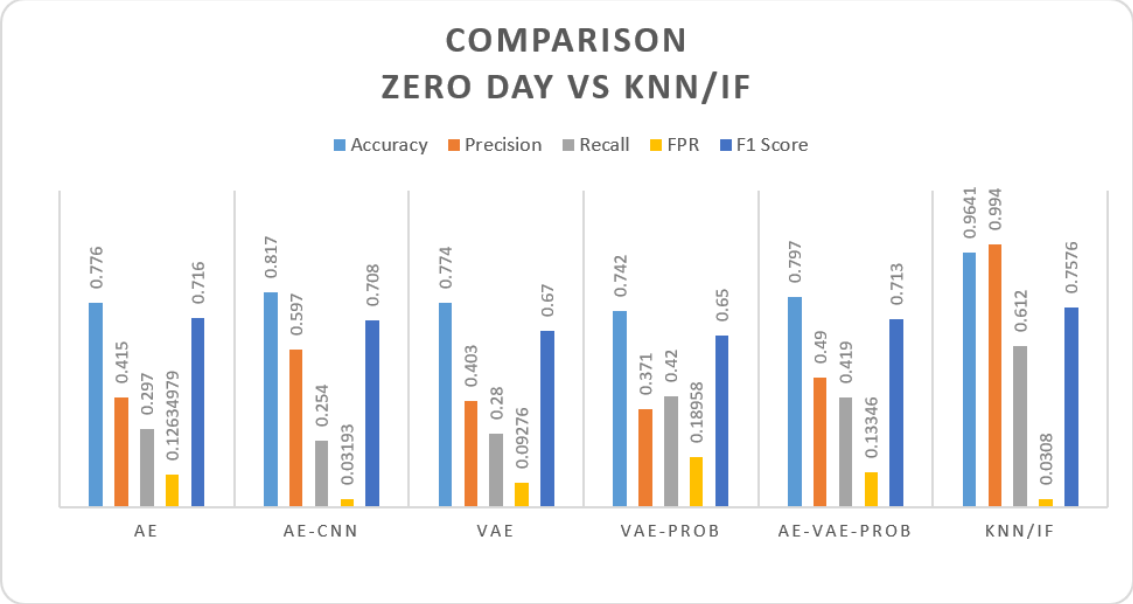
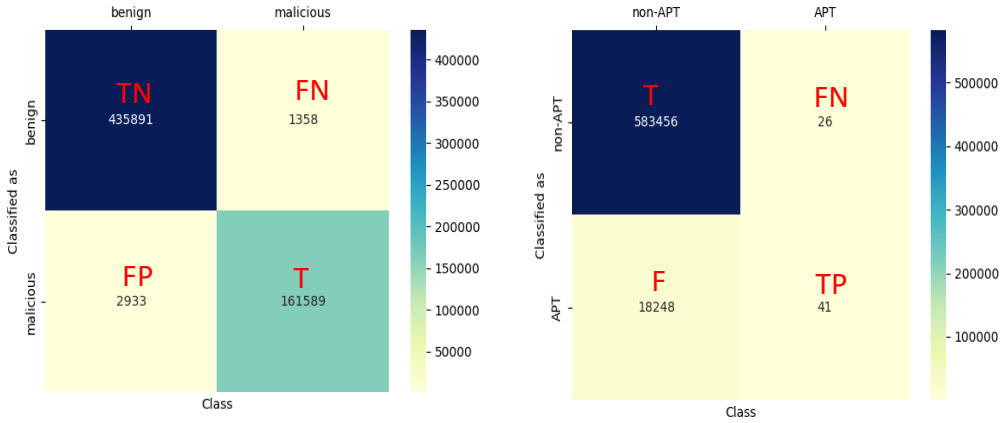


Figure 16 Performance Metrics Comparison of KNN/IF vs. Benchmark Paper (zero Day Detection)



Confusion Matrix of KNN (first stage)

Confusion Matrix of Isolation Forest (second stage)

Figure 17: Confusion Matrix for the 20% validation of 2-Stage APT Detection

The selected KNN-IF combination also generates the lowest false positive rate when compared to both referenced article's anomaly detection and zero day, by 0.2% and 2%, respectively (see Tables 15 and 16). When compared to other APT detection papers that use false positive rate as their metrics, the results for the final approach still outperformed those papers (see Table 17).

(Xuan, et al, 2021) cited 29 times A new approach for APT malware Detection based on deep graph Network for endpoint systems	
	False Positive Rate (FPR)
GCN Model	9.18%
GCN model	7.42%
DGCNN Model	8.00%
GIN Model	4.80%
Ghafir, et al, 2018) cited 176 times Detection of Advanced Persistent Threat Using Machine Larning Correlation Analysis	
	False Positive Rate (FPR)
MLAPT	4.50%
TerminAPTor	high
Spear Phising based	14.20%
Context-based	27.88%
(Abdullayeva, 2021) Cited 13 times Advanced Persistent Threat attack deteciton method in cloud computing based on autoencoder and softmax regression algorithm	
	False Positive Rate (FPR)
Autoencoder	45.15%
(Niu et al, 2017) Identifying APT Malware Domain Based on Mobile DNS Logging	
	False Positive Rate (FPR)
IForest	5.20%
LOF	16.90%
KNN	20%
GAF	5.60%
(Xuan, 2021) Detection APT attacked Based on Network Traffic Using Machine Learning	
	False Positive Rate (FPR)
RF(50 trees)	3.72%
RF(30 treees)	4.36%
RF (100 trees)	4.15%

Table 17: APT Detection False Positive Rate Reference Articles Review

Chapter 5—Discussion and Conclusions

In this praxis, a two-stage classification/detection model was developed to detect and classify APT attacks on semi-synthetic datasets. The algorithms used in the two stages include KNN for the first stage, and the unsupervised algorithm Isolation Forest for the second stage. The experimental results in Chapter 4 have shown that the proposed two–stage classification/detection model resulted in significant accuracy and precision, with a low false positive rate and high F1-score in detecting APT attacks.

5.1 Discussion

The praxis began utilizing semi-synthetic datasets that mimic real-world network traffic and malware using both CICIDS2017 data and APT PCAP files from the Contagio database. The combination of these two databases created by Dr. Stojanovic from Joanneum Research, and in the referenced Journal Article Neuschmied et al., 2022, enabled this praxis to benchmark performance metrics with regard to Anomaly Detection and Zero-Day Attack Detection against the referenced paper. Tables 15 and 16 illustrate the improvement this praxis achieved over the benchmark reference paper. Features unique to APT attacks were selected using WEKA functions. Given that APT attacks are typically well hidden and persistent, identifying their specific features has not only improved efficiency in the model training time, but has also improved detection accuracy. This praxis was able to use a smaller number of features compared to other APT detection papers while still producing higher detection accuracy. The proposed two-stage classification/detection model with supervised and unsupervised ML models has shown that the algorithms can detect not only malicious attacks, but also specific APT attacks. Furthermore, the model achieved greater accuracy in detecting true APT attacks against the benchmark paper that could not do the same.

The biggest challenge with respect to the experiments concerned the size of the dataset. In particular, a large dataset was used with few APT attacks (only 300 APT attacks out of approximately 3 million data records), and the highly imbalanced dataset initially caused overfitting with three labeled datasets for all ML algorithms. With the two-stage model, the data size was reduced after the first stage, with only 10% of overall data used for testing, and 20% of overall data used for validation.

5.2 Conclusions

This praxis achieved a significant goal of identifying a two-stage ML model to improve early detection of APT attacks. The praxis also fulfilled several other objectives, including the identification of features unique to APT attacks, and the successful use of those features when training the various ML/DL supervised and unsupervised models. However, the praxis also encountered certain setbacks. The initial strategy of identifying a specific ML or DL supervised algorithm to improve APT attack detection failed due to imbalances between benign network records, non-APT attacks and APT attack records in the dataset.

In pursuit of better precision and recall metrics, and to address the imbalanced datasets concern, this praxis proposes a two-stage classification/detection model. Following the first failed experiment, a subsequent achievement was demonstrating that KNN, DT and NB outperformed other algorithms (including DL algorithms), particularly with respect to training time and testing time. At the start of the second experiment for the two-stage model, all three of the algorithms (KNN, DT and NB) were used for both stages, as in one classifier on top of another. The end result for the second stage was inconclusive. To further address the inconclusive metrics, several unsupervised ML

algorithms were reviewed for the second stage detection—in particular, the Isolation Forest that can handle large imbalanced datasets and is ideal for identifying outliers from the predicted non-APT malicious attacks. Given the algorithm's linear time complexity and low memory requirement, the training and testing time are significantly shorter and resulted in improved accuracy when detecting APT attacks. For KNN and Isolation Forest combined, training time was around 9 seconds, and for testing time, because of KNN, the algorithm took 120 seconds to complete. The benchmark paper did not provide time for testing, and therefore a comparison to testing time was not possible. During the experiment of the two-stage model, both KNN and DT were trained at stage one, and KNN demonstrated higher accuracy and precision over DT when predicting all malicious attacks. For the second stage, the isolation forest outperformed other unsupervised ML algorithms, and was able to further perform outlier detection of APT attacks within the predicted malicious attacks. The experimental results show that the proposed two-stage classification/detection model can detect anomaly attacks and APT attacks with improved accuracy, even with a large imbalance dataset.

The final results indicate that the proposed model achieved improvements of 5% in accuracy for anomaly detection, and over 16% in accuracy for zero-day detection against the benchmark paper. A larger improvement in precision and recall was achieved, ranging from 20% up to 50% (Tables 15 and 16). The false positive rate obtained from the two-stage model was also shown to be the lowest amongst some of the APT detection papers (Table 17). In addition, while much of the research discusses APT attacks, it does not differentiate between anomaly detection and APT attack detection, which is a significant shortcoming in this field. The proposed model expands on this research to

detect specific APT attacks within the anomalies, with significantly high accuracy and precision, and low false positives. When compared to other research in this field, the proposed model uses less features, but still yields better performance with shorter training and testing time.

5.3 Contributions to Body of Knowledge

This praxis has identified nine APT-specific features for ML/DL training and testing, which is critical to ensuring higher accuracy of APT attack detection. This contribution to the literature significantly reduces the processing time required to train ML/DL algorithms. Other work, such as the Maseers paper (Maseer et al., 2021) and the Hofer-Schmitz paper (Hofer-Schmitz et al., 2021), identify 35 and 19 features respectively, which lengthen the time to train these models and reduces the accuracy in detecting APT attacks.

The proposed two-stage classification/detection model is another significant contribution that has improved detection of anomaly attacks and APT attacks. This model also reduces the training and testing time windows previously demonstrated within the literature. APT detection research work has leveraged CNN, RNN, Autoencoders, KNN and softmax regression, but a combination of the KNN and the Isolation Forest has not previously been applied. The proposed model uses a supervised ML model KNN for the first stage, and uses an unsupervised ML model Isolation Forest for the second stage, to constitute the two-stage classification/detection model, which delivers higher detection accuracy and shorter training time and lower false positive rate.

5.4 Recommendations for Future Research

Few APT datasets are currently available in the research field. Of those available, some have used NSL-KDD, which has many redundancies and flaws, and others used

datasets such as CICIDS2017 / CICIDS2018, which does not include real-world APT attacks. Many referenced papers on APT detection actually performed anomaly detection with APT (not true APT detection), or claim to use APT data in the disclosed models, but without using actual APT attacks. Other papers have used detecting normal (benign) as true positive to show the high accuracy and precision with much lower APT and anomaly detection metrics. A real-world APT dataset from any high value enterprise networks would be very useful in benchmarking various performance metrics, and for future APT detection evaluation. Given that the unsupervised ML algorithm actually performed well in predicting APT attacks without compromising on training and testing time, further research could apply other unsupervised ML algorithms in addition to the three algorithms evaluated in this praxis. Another benefit of using unsupervised ML algorithms is their ability to train using imbalanced datasets. By comparison, imbalanced datasets have caused overfitting problems when used with supervised ML algorithms. Therefore, future research opportunities exist to investigate how imbalanced datasets might be addressed under different use cases, such as use with supervised ML algorithms. Another possible area of future research on APT would be an analysis of APT attacks at different stages, which this praxis has not covered. Detection of APT attacks at their initiation, onwards to escalation and through to their persistence has not been reviewed in detail; instead, review has typically focused only on network flow using features that are unique to APT attacks. Thus, there is ample scope for future research and improvement in APT attack detection.

References

1. Informa PLC Informa UK Limited. (2023, January 05). *Check Point Research Reports a 38% Increase In 2022 Global Cyberattacks.* https://www.darkreading.com/attacks-breaches/check-point-research-reports-a-38-increase-in-2022-global-cyberattacks
2. Khaleefa, E. J., & Abdulah, D. A. (2022). Concept and difficulties of advanced persistent threats (APT): Survey. *International Journal of Nonlinear Analysis and Applications*, *13*(1), 4037-4052.
3. Neuschmied, H., Winter, M., Stojanović, B., Hofer-Schmitz, K., Božić, J., & Kleb, U. (2022). APT-Attack Detection Based on Multi-Stage Autoencoders. *Applied Sciences*, *12*(13), 6816.
4. Maseer, Z. K., Yusof, R., Bahaman, N., Mostafa, S. A., & Foozy, C. F. M. (2021). Benchmarking of machine learning for anomaly based intrusion detection systems in the CICIDS2017 dataset. *IEEE access*, *9*, 22351-22370.
5. Liu, H., & Lang, B. (2019). Machine learning and deep learning methods for intrusion detection systems: A survey. *applied sciences*, *9*(20), 4396.
6. DeVore, M. R., & Lee, S. (2017). APT (advanced persistent threat) s and influence: Cyber weapons and the changing calculus of conflict. *The Journal of East Asian Affairs*, 39-64.
7. Jamali, N., & O'Connor, T. (2020). US, China's cold war is raging in cyberspace, where intellectual property is a costly front.
8. Brown, M., & Pavneet S., (2018). China's Technology Transfer Strategy: How Chinese Investments in Emerging Technology Enable A Strategic Competitor to Access the Crown Jewels of U.S. Innovation, U.S. Department of Defense, Defense Innovation Unit Experimental, January 2018.
9. Ghafir, I., & Prenosil, V. (2016). Proposed approach for targeted attacks detection. In *Advanced Computer and Communication Engineering Technology: Proceedings of ICOCOE 2015* (pp. 73-80). Springer International Publishing.
10. Tankard, C. (2011). Advanced persistent threats and how to monitor and deter them. *Network security*, *2011*(8), 16-19.

11. Binde, B., McRee, R., & O'Connor, T. J. (2011). Assessing outbound traffic to uncover advanced persistent threat. *SANS Institute. Whitepaper*, *16*.
12. Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). A survey of network-based intrusion detection data sets. *Computers & Security*, *86*, 147-167.
13. Sommer, R., & Paxson, V. (2010, May). Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy* (pp. 305-316). IEEE.
14. Mahdavifar, S., & Ghorbani, A. A. (2019). Application of deep learning to cybersecurity: A survey. *Neurocomputing*, *347*, 149-176.
15. Stojanović, B., Hofer-Schmitz, K., & Kleb, U. (2020). APT datasets and attack modeling for automated detection methods: A review. *Computers & Security*, *92*, 101734.
16. Neuschmied, H., Winter, M., Stojanović, B., Hofer-Schmitz, K., Božić, J., & Kleb, U. (2022). APT-Attack Detection Based on Multi-Stage Autoencoders. *Applied Sciences*, *12*(13), 6816.
17. Hofer-Schmitz, K., Kleb, U., & Stojanović, B. (2021). The influences of feature sets on the detection of advanced persistent threats. *Electronics*, *10*(6), 704.
18. Myneni, S., Chowdhary, A., Sabur, A., Sengupta, S., Agrawal, G., Huang, D., & Kang, M. (2020). DAPT 2020-constructing a benchmark dataset for advanced persistent threats. In *Deployable Machine Learning for Security Defense: First International Workshop, MLHat 2020, San Diego, CA, USA, August 24, 2020, Proceedings 1* (pp. 138-163). Springer International Publishing.
19. Khaleefa, E. J., & Abdulah, D. A. (2022). Concept and difficulties of advanced persistent threats (APT): Survey. *International Journal of Nonlinear Analysis and Applications*, *13*(1), 4037-4052.
20. Alshamrani, A., Myneni, S., Chowdhary, A., & Huang, D. (2019). A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Communications Surveys & Tutorials*, *21*(2), 1851-1877.
21. Auty, M. (2015). Anatomy of an advanced persistent threat. *Network Security*, *2015*(4), 13-16.

22. Quintero-Bonilla, S., & Martín del Rey, A. (2020). A new proposal on the advanced persistent threat: A survey. *Applied Sciences*, *10*(11), 3874.
23. Lemay, A., Calvet, J., Menet, F., & Fernandez, J. M. (2018). Survey of publicly available reports on advanced persistent threat actors. *Computers & Security*, *72*, 26-59.
24. Mandiant. APT1 Exposing One of China's Cyber Espionage Units; Technical report; Mandiant: Alexandria, VA, USA, 2013.
25. Salahdine, F., & Kaabouch, N. (2019). Social engineering attacks: A survey. *Future Internet*, *11*(4), 89.
26. Parmar, B. (2012). Protecting against spear-phishing. *Computer Fraud & Security*, *2012*(1), 8-11.
27. Ismail, K. A., Singh, M. M., Mustaffa, N., Keikhosrokiani, P., & Zulkefli, Z. (2017). Security strategies for hindering watering hole cyber crime attack. *Procedia Computer Science*, *124*, 656-663.
28. Bilge, L., & Dumitraş, T. (2012, October). Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security* (pp. 833-844).
29. Yang, L. X., Li, P., Yang, X., & Tang, Y. Y. (2017). Security evaluation of the cyber networks under advanced persistent threats. *IEEE access*, *5*, 20111-20123.
30. John, J. T. (2017). State of the art analysis of defense techniques against advanced persistent threats. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communication (IITM) Focal Topic: Advanced Persistent Threats*, *63*.
31. Do Xuan, C., Dao, M. H., & Nguyen, H. D. (2020). APT attack detection based on flow network analysis techniques using deep learning. *Journal of Intelligent & Fuzzy Systems*, *39*(3), 4785-4801.
32. Chu, W. L., Lin, C. J., & Chang, K. N. (2019). Detection and classification of advanced persistent threats and attacks using the support vector machine. *Applied Sciences*, *9*(21), 4579.
33. Virvilis, N., & Gritzalis, D. (2013, September). The big four-what we did wrong in advanced persistent threat detection?. In *2013 international conference on availability, reliability and security* (pp. 248-254). IEEE.

34. Omar, S., Ngadi, A., & Jebur, H. H. (2013). Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications*, *79*(2).
35. Skopik, F., Settanni, G., Fiedler, R., & Friedberg, I. (2014, July). Semi-synthetic data set generation for security software evaluation. In *2014 Twelfth Annual International Conference on Privacy, Security and Trust* (pp. 156-163). IEEE.
36. Choobdar, P., Naderan, M., & Naderan, M. (2022). Detection and multi-class classification of intrusion in software defined networks using stacked auto-encoders and CICIDS2017 dataset. *Wireless Personal Communications*, 1-35.
37. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, *1*, 108-116.
38. McHugh, J. (2000). Testing inturstion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by Lincoln laboratory. *ACM Trans. Inf. Syst. Secur*., 3(4): 262-294.
39. Bodström, T., & Hämäläinen, T. (2019). A novel deep learning stack for APT detection. *Applied Sciences*, *9*(6), 1055.
40. Shenfield, A., Day, D., & Ayesh, A. (2018). Intelligent intrusion detection systems using artificial neural networks. *Ict Express*, *4*(2), 95-99.
41. Somvanshi, M., Chavan, P., Tambade, S., & Shinde, S. V. (2016, August). A review of machine learning techniques using decision tree and support vector machine. In *2016*
42. *international conference on computing communication control and automation (ICCUBEA)* (pp. 1-7). IEEE.
43. Martínez Torres, J.; Iglesias Comesaña, C.; García-Nieto, P.J. Review: machine learning techniques applied to cybersecurity. *Int. J. Mach. Learn. Cybern*. 2019, 10, 2823-2836
44. Gao, B., Ma, H. Y., & Yang, Y. H. (2002, November). Hmms (hidden markov models) based on anomaly intrusion detection method. In *Proceedings. International Conference on Machine Learning and Cybernetics* (Vol. 1, pp. 381-385). IEEE.
45. Siddiqui, S., Khan, M. S., Ferens, K., & Kinsner, W. (2016, March). Detecting advanced persistent threats using fractal dimension based machine learning classification. In *Proceedings of the 2016 ACM on international workshop on security and privacy analytics* (pp. 64-69).

46. Janiesch, C., Zschech, P. & Heinrich, K. Machine learning and deep learning. *Electron Markets* **31**, 685–695 (2021).
47. Janiesch, C., Zschech, P. & Heinrich, K. Machine learning and deep learning. *Electron Markets* **31**, 685–695 (2021). https://doi.org/10.1007/s12525-021-00475-2
48. Berman, D. S., Buczak, A. L., Chavis, J. S., & Corbett, C. L. (2019). A survey of deep learning methods for cyber security. *Information*, *10*(4), 122.
49. Abdullayeva, F. J. (2021). Advanced persistent threat attack detection method in cloud computing based on autoencoder and softmax regression algorithm. *Array*, *10*, 100067.
50. Min, B., Yoo, J., Kim, S., Shin, D., & Shin, D. (2021). Network anomaly detection using memory-augmented deep autoencoder. *IEEE Access*, *9*, 104695-104706.
51. Zhang, J., Pan, L., Han, Q. L., Chen, C., Wen, S., & Xiang, Y. (2021). Deep learning based attack detection for cyber-physical system cybersecurity: A survey. *IEEE/CAA Journal of Automatica Sinica*, *9*(3), 377-391.
52. Zhao, D., Liu, J., Wang, J., Niu, W., Tong, E., Chen, T., & Li, G. (2019). Bidirectional RNN-based few-shot training for detecting multi-stage attack. *arXiv preprint arXiv:1905.03454*.
53. Do Xuan, C., Dao, M. H., & Nguyen, H. D. (2020). APT attack detection based on flow network analysis techniques using deep learning. *Journal of Intelligent & Fuzzy Systems*, *39*(3), 4785-4801.
54. Do Xuan, C., & Duong, D. (2022). Optimization of APT attack detection based on a model combining ATTENTION and deep learning. *Journal of Intelligent & Fuzzy Systems*, *42*(4), 4135-4151.
55. Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., ... & Wang, C. (2018). Machine learning and deep learning methods for cybersecurity. *Ieee access*, *6*, 35365-35381.
56. Frank, E., & Bouckaert, R. R. (2006). Naive bayes for text classification with unbalanced classes. In *Knowledge Discovery in Databases: PKDD 2006: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases Berlin, Germany, September 18-22, 2006 Proceedings 10* (pp. 503-510). Springer Berlin Heidelberg.

57. Rennie, Jason D., et al. "Tackling the poor assumptions of naive bayes text classifiers." *Proceedings of the 20th international conference on machine learning (ICML-03)*. 2003.
58. Yijing, L., Haixiang, G., Xiao, L., Yanan, L., & Jinling, L. (2016). Adapted ensemble classification algorithm based on multiple classifier system and feature selection for classifying multi-class imbalanced data. *Knowledge-Based Systems*, *94*, 88-104.

Appendix A

Figure 1: Code to Reformat Datasets for WEKA

.CSV clean up

toCSV.py

```
import csv

oriFile = input("The file name to convert: ")
fileType = input("File type: ")

file1 = open(oriFile + "." + fileType, 'r')
lines = file1.readlines()
rows = []
fields = []

cnt = 0
for line in lines:
    cnt += 1
    if cnt == 1:
        line = line.strip('\n')
        fields.extend(line.split(';'))
        print("fields: ", fields)
    else:
        line = line.strip('\n')
        tmpList = line.split(';')
        rows.append(tmpList)

file1.close()

print("rows:")
print(rows)

with open(oriFile + "_new." + fileType, 'w') as file2:
    write = csv.writer(file2)
    write.writerow(fields)
    write.writerows(rows)


print("Done.")
print("orginal file: ", oriFile + "." + fileType)
print("converted file: ", oriFile + "_new." + fileType)
```

Figure 2: Sample Feature Selection

CIC_thu_1150CleannedLabelled.csv

Full training set

Instances: 458980
Attributes: 75

Evaluator: weka.attributeSelection.ClassifierAttributeEval -execution-slots 1 -B weka.classifiers.rules.ZeroR -F 5 -T 0.01 -R 1 -E DEFAULT --
Search: weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1

Ranked attributes:

0 74 Total Length of Fwd Packet

0 24 Dst IP

0 26 FIN Flag Count

0 25 Dst Port

0 23 Down/Up Ratio

0 19 Bwd Packet Length Std

0 22 Bwd Segment Size Avg

0 21 Bwd Packets/s

0 27 FWD Init Win Bytes

0 28 Flow Bytes/s

0 29 Flow Duration

0 30 Flow IAT Max

0 35 Flow Packets/s

0 34 Flow ID

0 33 Flow IAT Std

0 32 Flow IAT Min

0 31 Flow IAT Mean

0 20 Bwd Packet/Bulk Avg

0 18 Bwd Packet Length Min

0 73 Total Length of Bwd Packet

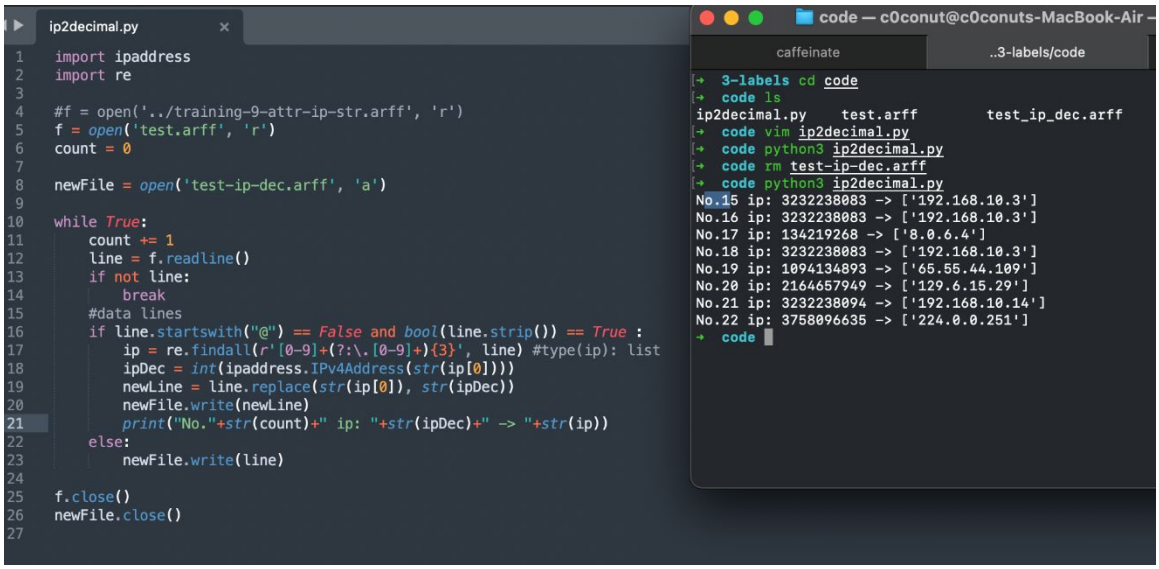
0 5 Active Std

0 7 Bwd Bulk Rate Avg

0 6 Average Packet Size
0 4 Active Min
0 17 Bwd Packet Length Mean
0 3 Active Mean
0 2 Active Max
0 8 Bwd Bytes/Bulk Avg
0 9 Bwd Header Length
0 10 Bwd IAT Max
0 11 Bwd IAT Mean
0 16 Bwd Packet Length Max
0 15 Bwd Init Win Bytes
0 14 Bwd IAT Total
0 13 Bwd IAT Std
0 12 Bwd IAT Min
0 36 Fwd Act Data Pkts
0 37 Fwd Header Length
0 38 Fwd IAT Max
0 61 Protocol
0 63 SYN Flag Count
0 62 RST Flag Count
0 60 Packet Length Variance
0 39 Fwd IAT Mean
0 59 Packet Length Std
0 58 Packet Length Min
0 64 Src IP
0 65 Src Port
0 66 Subflow Bwd Bytes
0 67 Subflow Bwd Packets
0 72 Total Fwd Packet
0 71 Total Bwd packets
0 70 Timestamp

0 69 Subflow Fwd Packets
0 68 Subflow Fwd Bytes
0 57 Packet Length Mean
0 56 Packet Length Max
0 55 PSH Flag Count
0 46 Fwd Packet Length Min
0 44 Fwd Packet Length Max
0 43 Fwd PSH Flags
0 42 Fwd IAT Total
0 41 Fwd IAT Std
0 40 Fwd IAT Min
0 45 Fwd Packet Length Mean
0 47 Fwd Packet Length Std
0 54 Idle Std
0 48 Fwd Packets/s
0 53 Idle Min
0 52 Idle Mean
0 51 Idle Max
0 50 Fwd Segment Size Avg
0 49 Fwd Seg Size Min
0 1 ACK Flag Count
top 5:
0 74 Total Length of Fwd Packet
0 24 Dst IP
0 26 FIN Flag Count
0 25 Dst Port
0 23 Down/Up Ratio

Figure 3: IP value to Decimal Conversion for WEKA



Appendix B

Approach One Configuration and Results

The first round of experiments was to detect APT attacks using three label classifications: benign, APT and attacks using various ML/DLM algorithms. The algorithms include NaïveBayes, Decision Tree J48, K Nearest Neighbors (KNN), Support Vector Machine (SVM), Neural Networks (NN), Convolution Neural Network (CNN) and Recurrent Neural Network (RNN). The results and summary of the ML/DL algorithms are provided below.

The following configurations for the algorithms were applied:

- NaïveBayes: NB default; batch size 32; batch size 32 useSupervised Discretization; batch size 32 useKernelEstimator; batch size 64 useSupervised Discretization; batch size 64 useKernelEstimator; batch size 64; batch size 128; batch size 256;
- Decision Tree J48: confidenceFactor 0.05minNumObj2 with batch_size 32, batch size 64 and batch size 128; confidenceFactor 0.1minNumObj2 with batch_size 32; confidenceFactor 0.2minNumObj2 with batch_size 32; confidenceFactor 0.25minNumObj2 with batch_size 32; confidenceFactor 0.3minNumObj2 with batch_size 32; confidenceFactor 0.05minNumObj3 with batch_size 64; confidenceFactor 0.05minNumObj4 with batch_size 64; confidenceFactor 0.05minNumObj5 with batch_size 64;
- KNN;
- SVM: C_SVM+polynomial Kernel; and
- Neural network.

For deep learning algorithms, TensorFlow 2.9.1 GPU with Kera was used with Python3 for:

- CNN; and
- RNN.

TensorFlow is an end-to-end open source platform for ML that uses TF API to develop and train ML models. Keras is a library containing high-level neural network that runs on top of TF.

In the first round using WEKA and the three labels (benign, attacks and APTs), the original expectation was to compare these selected algorithms for the best performance of detecting APT attacks; however, most results from the algorithms were inconclusive.

First Round Evaluation Results

NaïveBayes Summary:

NaïveBayes was run first under WEKA. Table 1 shows the consolidated metrics for the various runs carried out using useKernelEstimator and useSupervisedDiscretization parameters ranging from batch size 32 through 256. UseKernelEstimator uses a kernel estimator for numeric attributes rather than a normal distribution, and use Supervised-Discretization to convert numeric attributes to nominal ones in case one needs to discretize the data. Discretization is defined as a process of converting continuous data attribute values into a finite set of intervals with minimal loss of information. For APT detection, when batch size[7] is set at 128, the results demonstrate

[7] "Batch size" refers to the number of training examples used in one iteration, or number of samples that propagate through the network before updating the mode parameters.

the best training and testing time amongst all other batch sizes. The precision is at .02% with accuracy of 87.6% and recall of 64.3%.

batch size	useKernelEstimator	useSupervisedDiscretization	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	raining time	esting time
32	N	N	0.643	0.123	0.000	0.643	0.000	0.009	0.884	0.001	2.25	2.92
32	N	Y	0.000	0.000	0.000	0.000	0.000	-0.000	0.753	0.001	11.38	1.76
32	Y	N	0.000	0.000	0.000	0.000	0.000	-0.000	0.865	0.001	2.74	130.3
64	N	N	0.643	0.123	0.000	0.643	0.000	0.009	0.884	0.001	2	2.96
64	N	Y	0.000	0.000	0.000	0.000	0.000	-0.000	0.753	0.001	11.2	1.42
64	Y	N	0.000	0.000	0.000	0.000	0.000	-0.000	0.753	0.001	11.2	1.42
128	N	N	0.643	0.123	0.000	0.643	0.000	0.009	0.884	0.001	1.83	2.98
256	N	N	0.643	0.123	0.000	0.643	0.000	0.009	0.884	0.001	1.83	3.07

Table 1: Performance Metrics for the various Batch Size for NaiveBayes

Decision Tree J48 Summary:

Table 2 below summarizes the metrics for various combinations of confidence factors ranging from 0.05 to 0.3, with batch size ranging from 32 to 128, and minimum number of objects ranging from 2 to 5 for the Decision Tree J48 algorithm. The decision tree algorithm is a classification algorithm that produces decision trees with a top-down recursive, divide-and-conquer strategy. The confidence level determines how aggressive the pruning process will be; the higher it is, the more confidence there will be that the dataset is a complete representation of all possible events. The number of objects is the minimum number of observations allowed at each leaf of the tree. The tree size has 37 leaves below. The parameters that performed the best were 128 batches at a confidence level of 0.05 and a number of objects at 2.

The precision and accuracy for the best performed 128 batches algorithm are at 66.7% and 99.9%, respectively. Even though the data demonstrates high accuracy and

precision, the recall number of 14.3% indicates an outcome of an imbalanced class of datasets.

Batch Size	Number of Objects	Confidence Level	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	training time	testing time
32	2	.05	0.143	0.000	0.667	0.143	0.235	0.309	0.449	0.095	21.18	0.83
32	2	.10	0.143	0.000	1.000	0.143	0.250	0.378	0.416	0.143	18.94	0.74
32	2	0.20	0.143	0.000	1.000	0.143	0.250	0.378	0.416	0.143	19.78	0.91
32	2	0.25	0.143	0.000	1.000	0.143	0.250	0.378	0.416	0.143	19.43	0.73
32	2	0.30	0.143	0.000	0.667	0.143	0.235	0.309	0.449	0.095	19.08	0.81
64	2	0.05	0.143	0.000	0.667	0.143	0.235	0.309	0.449	0.095	19.5	0.7
128	2	0.05	0.143	0.000	1.000	0.143	0.250	0.378	0.416	0.143	19.0	0.74
64	3	0.05	0.143	0.000	1.000	0.143	0.250	0.378	0.416	0.143	19.96	1.01
64	4	0.05	0.143	0.000	1.000	0.143	0.250	0.378	0.416	0.143	19.55	0.79
64	5	0.05	0.143	0.000	1.000	0.143	0.250	0.378	0.416	0.143	19.49	0.78

Table 2: Performance Metrics for Decision Tree J48

KNN Summary:

For KNN, the algorithm uses proximity (i.e., similar things that are near to each other) to make predictions. The results shown below are not very conclusive. For a KNN with k = 5 (nearest neighbors), training time was 0.23 seconds, but the testing time took much longer at 58684.74 seconds (approximately 16 hours). Given the True Positive (TP) was at zero for APT detection, both precision and recall would be zero.

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	0.605	0.981	1.000	0.990	0.616	0.696	0.977	benign
	0.396	0.000	0.981	0.396	0.564	0.617	0.696	0.441	attack
	0.000	0.000	0.000	0.000	0.000	0.000	0.494	0.002	APT
Weighted Avg.	0.981	0.586	0.981	0.981	0.977	0.616	0.696	0.961	

Table 3: Performance Metrics for KNN

SVM Summary:

For SVM, the algorithm created a line or hyperplane to separate data into classes or categories. Sequential minimal optimization (SMO) used to solve quadratic programming problems during the training of SVM was applied, but it would not complete training regardless of what kernel was used. NU-SVM outputted errors and C-SVM would only work with Poly Kernel, but the data was inclusive with zero precision and F-measures. Therefore, SVM has been excluded from the evaluation.

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.000	0.000	?	0.000	?	?	0.500	0.969	benign
	0.000	0.000	?	0.000	?	?	0.500	0.031	attack
	1.000	1.000	0.000	1.000	0.000	?	0.500	0.000	APT
Weighted Avg.	0.000	0.000	?	0.000	?	?	0.500	0.940	

Table 4: Performance Metrics for SVM

Neural Network (NN) Summary:

As compared to the other ML algorithms, neural network had the longest training time at 5,300.33 seconds (approximately 88 minutes), while taking only 4.93 seconds to test. For Sigmoid function, nodes 0-8 were tested with NN as the activation function, which allows the network to introduce nonlinearity into the model. The same was done with KNN, where the TP was at 0 for detecting APT, which would not provide any useful measurement for Accuracy and Precision.

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	1.000	0.969	1.000	0.984	-0.003	0.798	0.993	benign
	0.000	0.000	0.000	0.000	0.000	-0.003	0.470	0.028	attack
	0.000	0.000	?	0.000	?	?	0.762	0.000	APT

Weighted Avg.	0.969	0.969	?	0.969	?	?	0.787	0.963	

Table 5: Performance Metrics for NN

CNN Summary:[8]

For both deep learning algorithms CNN and RNN, Tensorflow (TF) and Keras were used for model fitting and prediction and SKLEAN was used for evaluation. The categorical crossentropy loss was also calculated for the multi-class classification. With the loss, the CNN can be trained to output a probability over the three classes (benign, attacks and APTs). TF has built-in methods to train, evaluate and test the CNN algorithm. The model was trained with a model.fit() function (with validation_split = 0.10) and saved when loss function was improved. Following this, evaluate() and predict() functions were used for testing the testing dataset. The function evaluate() predicts the output for a given input, computes the metrics function specified in model.compile, and, based on y_true and y_pred, returns the computed metric value as the output and outputs the loss and accuracy values. Predict() only returns the y-pred, which is the prediction of y(label) according to the given x that is used to compute the confusion matrix after comparing to the real label.

The adaptive deep neural network training optimizer (Adam), and the stochastic gradient descent (SGD) optimizer, are both used for CNN. With Adam, the training time is about 3363.3 seconds (approximately 56 minutes) and the testing time is about 9.7 seconds. Table 6 below shows inconclusive metrics for the algorithm's performance metrics.

[8] See Appendix B Figure 1 for code.

	Precision	Recall	F1-score	Support
APT	0.0000	0.0000	0.0000	14
Attack	0.0000	0.0000	0.0000	13836
Benign	0.9689	1.0000	0.9842	432072

Table 6: Performance Metrics for CNN-Adam

With the SGD optimizer, the training time is 3068.3 seconds, which is moderately improved when compared to the Adam with a testing time of about 10 seconds. Table 7 has also shown no conclusive performance metrics.

	Precision	Recall	F1-score	Support
APT	0.0000	0.0000	0.0000	14
Attack	0.0000	0.0000	0.0000	13836
Benign	0.9689	1.0000	0.9842	432072

Table 7: Performance Metrics for CNN-SGD

RNN Summary:[9]

With the Adam's optimizer, the training time was 7208.7 seconds (approximately 120 minutes) and the testing time was 20.18 seconds. With the SGD optimizer, the training time was 7165.6 seconds and the testing time was about 20.3 seconds. Both have the same confusion matrix and same accuracy of 96.68%.

	Precision	Recall	F1-score	Support

[9] See Appendix B Figure 2 for Code.

APT	0.0000	0.0000	0.0000	14
Attack	0.0000	0.0000	0.0000	13836
Benign	0.9689	.9977	0.9831	432072

Table 8: Performance Metrics for RNN

Both CNN and RNN have observably skewed precision and recall that are also reflected in the confusion matrix. The reason for this is because of the highly imbalanced and skewed datasets which result in biased predictions.

	Training time (sec)	**Testing time (sec)**
NB	1.83	2.98
DTJ48	19	0.74
KNN	0.23	58684.74
NN	5300.33	4.93
CNN (Adam)	3363.3	9.7
CNN (SGD)	3068.3	10
RNN (Adam)	7208.7	20.18
RNN(SGD)	7165.6	20.3

Table 9: Training and Testing time for the ML/DL Algorithms

Figure 1 below captured a summary of the performance of the algorithms. NB demonstrated high accuracy and recall but very low precision, which may indicate relatively good prediction with some high false positives at 12.3%. DTJ48 demonstrated high accuracy and precision but low recall, which is another indication of an imbalanced dataset for classification.

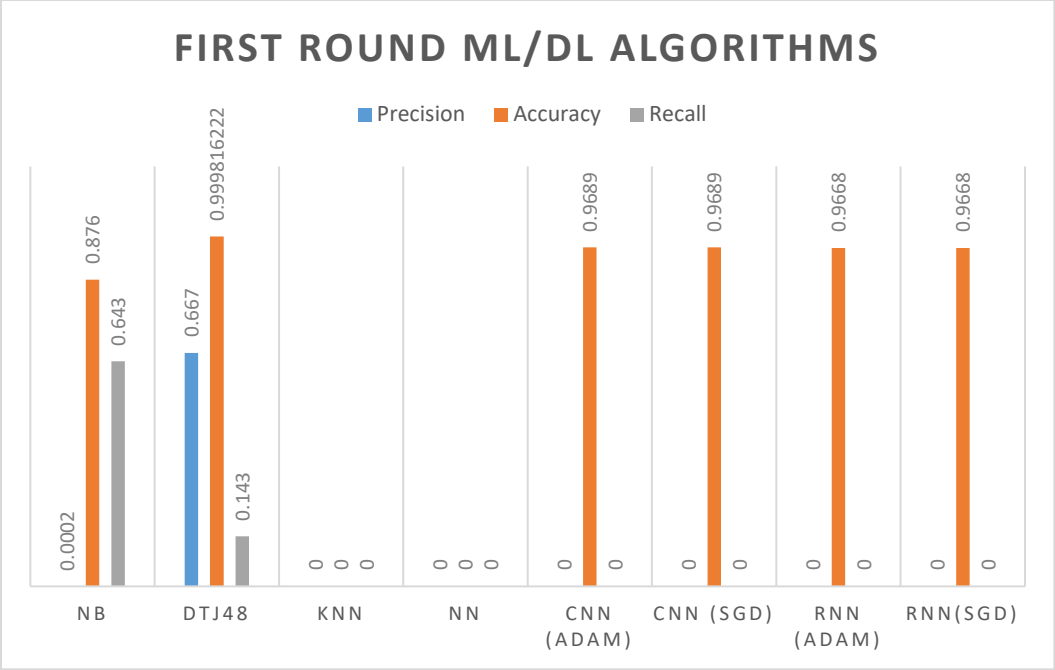


Figure 1: First Round of Experiments – Summary of Performance Metrics

Figure 2: CNN Python Code

```
import pandas as pd
import numpy as np
np.random.seed(2500)
import tensorflow as tf
from keras.utils import np_utils
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Input, Dropout, Flatten, Dense
from keras.layers import Conv1D, MaxPooling1D
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import optimizers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, f1_score
from sklearn.metrics import precision_score, recall_score
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sn
from laplotter import LossAccPlotter
from data_frequency import DataFrequency
import time

start_time = time.time()

#It is generally known that CNN is an effective method for image data

def plot_label_frequecy(label_frequency_distribution):
    label_frequency = sorted(label_frequency_distribution.items())
    keys = sorted(label_frequency_distribution.keys())
    df_label_freq = pd.DataFrame(label_frequency[7:], index=keys[7:], \
                                 columns=['class','frequency'])
    df_label_freq.plot.bar(figsize=(20,15), grid=True, fontsize=16, rot=30)
    plt.xlabel('Class', fontsize=20)
    plt.ylabel('Frequency (%)', fontsize=20)
    plt.yticks(np.arange(0.0, 10.05, 0.5), fontsize=20)
    plt.show()

train = pd.read_csv("dataset/training-9-attr.csv").values
x_train, y_train = [], []
for train_data in train:
    #features
    x_train.append(train_data[0:len(train_data)-1])
    #print(len(train_data[0:len(train_data)-1]))
```

```
for train_data in train:
    #features
    x_train.append(train_data[0:len(train_data)-1])
    #print(len(train_data[0:len(train_data)-1]))
    #print(train_data[0:len(train_data)-1])
    #class
    y_train.append(train_data[len(train_data)-1])
    #print("features:")
    #print(train_data[0:len(train_data)-2])
    #print("class: ", train_data[len(train_data)-1])
#convert to float32 for model fitting
x_train = np.array(x_train).astype(np.float32)
y_train = np.array(y_train)

data_freq = DataFrequency(y_train)
#print("data_freq: ", data_freq)
label_frequency_distribution = data_freq.calculate_label_distribution()
print("label_frequency_distribution: ", label_frequency_distribution)

test = pd.read_csv("dataset/testing.csv").values
x_test, y_test = [], []
for test_data in test:
    #features
    x_test.append(test_data[0:len(test_data)-1])
    #class
    y_test.append(test_data[len(test_data)-1])
    #print("features:")
    #print(test_data[0:len(test_data)-2])
    #print("class: ", test_data[len(test_data)-1])
#convert to float32 for model fitting
x_test = np.array(x_test).astype(np.float32)
y_test = np.array(y_test)

# reshape attributes to spatial dimensions
x_train = np.expand_dims(x_train, axis=2)
x_test = np.expand_dims(x_test, axis=2)

# one-hot-encode output labels
encoder = LabelEncoder()
encoder.fit(y_train)
encoded_y_train = encoder.transform(y_train)
y_train = np_utils.to_categorical(encoded_y_train)
encoder.fit(y_test)
# the name of the class labels encoded
```

```
# the name of the class labels encoded
class_labels = encoder.classes_
# the number of different labels being trained
nb_classes = len(class_labels)
encoded_y_test = encoder.transform(y_test)
y_test = np_utils.to_categorical(encoded_y_test)

# Build a model
model = Sequential()
model.add(Conv1D(filters=2, kernel_size=5, activation='relu', input_shape=(9,1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(nb_classes, activation='softmax'))
print(model.summary())

#train the model
#Experiment with different optimizers(i.e. Adam, RMSprop, SGD-Momentum, SGD-Nesterov)
#Set nb_epochs depending on the optimizer
optimizer = 'SGD-Nesterov'
print("optimizer: ", optimizer)
gd_optimizer = optimizers.SGD(learning_rate=0.001, momentum=0.9, nesterov=False)
nb_epochs = 50
print("nb_epochs: ", nb_epochs)
model.compile(optimizer=gd_optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# load model and weights from previous epochs
saved_model_file = 'models/trained_model_CNN_SGD-Nesterov.h5'
weights = 'models/weight_model_CNN_SGD-Nesterov.hdf5'
try:
    model.load_weights(weights)
except:
    pass

#save model at checkpoints when loss function improved
#keras check point
checkpoint = ModelCheckpoint(saved_model_file, monitor='val_loss', save_best_only=True, verbose=1)
fit_history = model.fit(x_train, y_train, epochs=nb_epochs, batch_size=32,
                        validation_split=0.10, callbacks=[checkpoint])
print("Save model.")
# save model weights reuse later
model.save_weights(weights)
```

```
#save model at checkpoints when loss function improved
#keras check point
checkpoint = ModelCheckpoint(saved_model_file, monitor='val_loss', save_best_only=True, verbose=1)
fit_history = model.fit(x_train, y_train, epochs=nb_epochs, batch_size=32,
                        validation_split=0.10, callbacks=[checkpoint])
print("Save model.")
# save model weights reuse later
model.save_weights(weights)
print("Save weights")

train_time = time.time()
print('Model training finished.')
print("Time of training the model: ", train_time - start_time, " s")

start_time = time.time()

#evaluation
model = load_model('models/trained_model_CNN_SGD-Nesterov.h5')
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)

evaluation_time = time.time()
msg = "\nModel evaluation finished\nLoss: {}\tAccuracy: {}".format(loss, accuracy)
print(msg)
print("Time of model evaluation: ", evaluation_time - start_time, " s")

def print_confusion_matrix(y_labels, preds, class_labels):

    y_true_labels = [np.argmax(t) for t in y_labels]
    y_preds_labels = [np.argmax(t) for t in y_labels]

    cm = confusion_matrix(y_true_labels, y_preds_labels)
    print("Confusion matrix:\n", cm)

preds = model.predict(x_test, batch_size=32, verbose=0)

print_confusion_matrix(y_test, preds, class_labels)

y_true_labels = [np.argmax(t) for t in y_test]
y_preds_labels = [np.argmax(t) for t in preds]

class_metric_report = classification_report(y_true_labels, y_preds_labels, target_names=class_labels, digits=4)
print(class_metric_report)
```

Figure 3: RNN Python Code

```
import pandas as pd
import numpy as np
np.random.seed(2500)
import tensorflow as tf
from keras.utils import np_utils
from keras.models import Sequential
from keras.models import load_model, Model
from keras.layers import Input, Dropout, Flatten, Dense
from keras.layers import SimpleRNN
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import optimizers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, f1_score
from sklearn.metrics import precision_score, recall_score
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sn
from laplotter import LossAccPlotter
from data_frequency import DataFrequency
import time

start_time = time.time()

# RNN

def plot_label_frequecy(label_frequency_distribution):
    label_frequency = sorted(label_frequency_distribution.items())
    keys = sorted(label_frequency_distribution.keys())
    df_label_freq = pd.DataFrame(label_frequency[7:], index=keys[7:], \
                                 columns=['class','frequency'])
    df_label_freq.plot.bar(figsize=(20,15), grid=True, fontsize=16, rot=30)
    plt.xlabel('Class', fontsize=20)
    plt.ylabel('Frequency (%)', fontsize=20)
    plt.yticks(np.arange(0.0, 10.05, 0.5), fontsize=20)
    plt.show()

train = pd.read_csv("dataset/training-9-attr.csv").values
x_train, y_train = [], []
for train_data in train:
    #features
    x_train.append(train_data[0:len(train_data)-1])
    #print(len(train_data[0:len(train_data)-1]))
```

```
    #print(len(train_data[0:len(train_data)-1]))
    #print(train_data[0:len(train_data)-1])
    #class
    y_train.append(train_data[len(train_data)-1])
    #print("features:")
    #print(train_data[0:len(train_data)-2])
    #print("class: ", train_data[len(train_data)-1])
#convert to float32 for model fitting
x_train = np.array(x_train).astype(np.float32)
y_train = np.array(y_train)

data_freq = DataFrequency(y_train)
#print("data_freq: ", data_freq)
label_frequency_distribution = data_freq.calculate_label_distribution()
print("label_frequency_distribution: ", label_frequency_distribution)

test = pd.read_csv("dataset/testing.csv").values
x_test, y_test = [], []
for test_data in test:
    #features
    x_test.append(test_data[0:len(test_data)-1])
    #class
    y_test.append(test_data[len(test_data)-1])
    #print("features:")
    #print(test_data[0:len(test_data)-2])
    #print("class: ", test_data[len(test_data)-1])
#convert to float32 for model fitting
x_test = np.array(x_test).astype(np.float32)
y_test = np.array(y_test)

# reshape attributes to spatial dimensions
x_train = np.expand_dims(x_train, axis=2)
x_test = np.expand_dims(x_test, axis=2)

# one-hot-encode output labels
encoder = LabelEncoder()
encoder.fit(y_train)
encoded_y_train = encoder.transform(y_train)
y_train = np_utils.to_categorical(encoded_y_train, 3)
encoder.fit(y_test)
# the name of the class labels encoded
class_labels = encoder.classes_
# the number of different labels being trained
nb_classes = len(class_labels)
```

```
encoder.fit(y_test)
# the name of the class labels encoded
class_labels = encoder.classes_
# the number of different labels being trained
nb_classes = len(class_labels)
encoded_y_test = encoder.transform(y_test)
y_test = np_utils.to_categorical(encoded_y_test, 3)

# Simple RNN
model = Sequential()
model.add(SimpleRNN(128, input_shape=(9, 1)))
model.add(Dense(nb_classes, activation='softmax'))
print(model.summary())
```

```
print(model.summary())

#train the model
#Experiment with different optimizers(i.e. Adam, RMSprop, SGD-Momentum, SGD-Nesterov)
#Set nb_epochs depending on the optimizer
optimizer = 'SGD-Nesterov'
print("optimizer: ", optimizer)
gd_optimizer = optimizers.SGD(learning_rate=0.001, momentum=0.9, nesterov=False)
nb_epochs = 50
print("nb_epochs: ", nb_epochs)
model.compile(optimizer=gd_optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# load model and weights from previous epochs
saved_model_file = 'models/trained_model_DVRNN_SGD_N.h5'
weights = 'models/weight_model_DVRNN_SGD_N.hdf5'
try:
    model.load_weights(weights)
except:
    pass

#save model at checkpoints when loss function improved
#keras check point
checkpoint = ModelCheckpoint(saved_model_file, monitor='val_loss', save_best_only=True, verbose=1)
fit_history = model.fit(x_train, y_train, epochs=nb_epochs, batch_size=32,
                        validation_split=0.10, callbacks=[checkpoint])
print("Save model.")
# save model weights reuse later
model.save_weights(weights)
print("Save weights")

train_time = time.time()
print('Model training finished.')
print("Time of training the model: ", train_time - start_time, " s")

start_time = time.time()

#evaluation
model = load_model('models/trained_model_DVRNN_SGD_N.h5')
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)

evaluation_time = time.time()
msg = "\nModel evaluation finished\nLoss: {}\tAccuracy: {}".format(loss, accuracy)
print(msg)
print("Time of model evaluation: ", evaluation_time - start_time, " s")
```

```
#save model at checkpoints when loss function improved
#keras check point
checkpoint = ModelCheckpoint(saved_model_file, monitor='val_loss', save_best_only=True, verbose=1)
fit_history = model.fit(x_train, y_train, epochs=nb_epochs, batch_size=32,
                        validation_split=0.10, callbacks=[checkpoint])
print("Save model.")
# save model weights reuse later
model.save_weights(weights)
print("Save weights")

train_time = time.time()
print('Model training finished.')
print("Time of training the model: ", train_time - start_time, " s")

start_time = time.time()

#evaluation
model = load_model('models/trained_model_DVRNN_SGD_N.h5')
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)

evaluation_time = time.time()
msg = "\nModel evaluation finished\nLoss: {}\tAccuracy: {}".format(loss, accuracy)
print(msg)
print("Time of model evaluation: ", evaluation_time - start_time, " s")

def print_confusion_matrix(y_labels, preds, class_labels):

    y_true_labels = [np.argmax(t) for t in y_labels]
    y_preds_labels = [np.argmax(t) for t in y_labels]

    cm = confusion_matrix(y_true_labels, y_preds_labels)
    print("Confusion matrix:\n", cm)

preds = model.predict(x_test, batch_size=32, verbose=0)

print_confusion_matrix(y_test, preds, class_labels)

y_true_labels = [np.argmax(t) for t in y_test]
y_preds_labels = [np.argmax(t) for t in preds]

class_metric_report = classification_report(y_true_labels, y_preds_labels, target_names=class_labels, digits=4)
print(class_metric_report)
```