Network Dataset Generation and Implementation of a Network Intrusion Detection System using Neural Networks

Fotios Dimitrios Tsokos

Supervisor: Prof. Papadimitriou Georgios

School of Informatics
Aristotle University of Thessaloniki

February 2021

In memoriam of my grandpa Fotis Tsokos,
I still recall your joy when you accompanied me to school for the first time.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Fotios Dimitrios Tsokos
February 2021

Acknowledgements

I would like to thank my supervisor, Professor Papadimitriou, for his faith in me, his support and advice. I would also want to thank Stefanos Papadopoulos for the lessons he offered me to expose me to the field of Neural Networks and Zack Stefanou for assisting me to convey Machine Learning in the best manner in the related chapter. Also this thesis would have been in a much worse situation without the support of Spyros Megalos, who despite his hectic schedule reviewed and offered revisions. Finally, I'd want to express my gratitude to my family for their unwavering support throughout the years, as well as Stella Lioka, for being a constant source of inspiration.

Περίληψη

Σε ένα διαρκώς μεταβαλλόμενο περιβάλλον, όπου νέες απειλές εμφανίζονται σε τακτική βάση, είναι ζωτικής σημασίας η δυνατότητα αναγνώρισης κακόβουλων πακέτων στο δίκτυο για την προστασία των χρηστών. Τα παραδοσιακά συστήματα ανίχνευσης εισβολών στο δίκτυο συγκρίνουν αποτυπώματα πακέτων για την ανίχνευση κακόβουλης δραστηριότητας, αλλά τα πιο σύγχρονα συστήματα χρησιμοποιούν μεθόδους μηχανικής μάθησης. Προκειμένου να εκπαιδευτούν στην ανίχνευση εισβολών, οι τελευταίες απαιτούν την ύπαρξη συνόλων δεδομένων. Ενώ έχει σημειωθεί τεράστια πρόοδος στην ανάπτυξη μοντέλων τεχνητής νοημοσύνης που ανιχνεύουν κακόβουλα πακέτα, το ζήτημα της παροχής των δεδομένων για την εκπαίδευσή τους υστερεί κατά πολύ. Στόχος της παρούσας διπλωματικής εργασίας είναι η συγγραφή λογισμικού που αναλύει δεδομένα πακέτων δικτύου και συμβουλεύεται ένα υπάρχον εργαλείο ανίχνευσης εισβολών για τη δημιουργία συνόλων δεδομένων από τα οποία μπορεί να εκπαιδευτεί ένα νευρωνικό δίκτυο για τον εντοπισμό εισβολών. Τα σύνολα δεδομένων που παράγονται, όπως και τα περισσότερα υπάρχοντα συστήματα, εστιάζουν στη ροή των πακέτων του δικτύου αλλά και στη δομή των επικεφαλίδων των πακέτων και στο περιεχόμενο των δεδομένων του.

Abstract

In an ever-changing environment where new threats emerge on a regular basis, it is critical to be able to recognize malicious packets on the network in order to safeguard users. Traditional network intrusion detection systems (NIDS) compare packet fingerprints to detect malicious activity, but more current systems use machine learning methods. In order to be trained in intrusion detection, the latter requires the presence of datasets. While tremendous progress has been made in the development of artificial intelligence models that detect malicious packets, the issue of providing the data to train them lags far behind. The goal of this thesis is to write software (Al Pastor) that analyzes network packet data (pcap files) and consults an existing intrusion detection tool (Snort) to generate datasets from which a neural network can be trained to identify intrusions. The produced datasets, like most existing systems, focus on the network's packet flow but also on the structure of the packet headers and their data content.

Contents

Introduction

Samuel Arthur paved the way for computer gaming and machine learning. He coined the term for the first time in 1959. It is defined as "the branch of science that enables computers to learn without being explicitly programmed."[1]. Since then, machine learning and artificial intelligence in general have progressed along previously unimaginable avenues.

In 1997, IBM's Deep Blue software[2] scored a significant victory against world chess champion Kasparov by using the Alpha-beta pruning approach, a branch-and-bound algorithm for solving NP-Hard problems. Although these approaches are not considered machine learning but rather a forerunner to artificial intelligence, they demonstrate that a computer may outperform a human champion on a certain topic. If the above is true for chess, which was a game of innovation until recently, then it may have comparable applicability in other industries.

The following example of a machine defeating a human occurs in the Chinese game of Go. DeepMind Technologies has developed software that uses the Monte Carlo technique to choose the optimal move from a tree of potential moves[3]. The tree was generated using machine learning and trained using data from several human-computer games.

As Isaac Asimov put it, "The most tragic element of existence at the moment is that science is accumulating information faster than society is accumulating wisdom". Perhaps we can alter the phrase to read as follows: "The most concerning but yet *encouraging* element of life at the moment is that machine learning accumulates information quicker than civilization accumulates wisdom".

The classical programming approach is characterized by severe staticity. To accommodate diverse data inputs, programmers may need to make adjustments. On the other hand, NPHard tasks are intractable (they can only be solved with techniques such as Branch & Bound). Because the programmer controls the program's control flow, the program's functioning is entirely dependent on the programmer's competence and expertise. However, the author of an algorithm, being human, cannot think of every possible scenario that might occur. This leads us to the conclusion that an unanticipated condition will always exist that might be devastating to the system and necessitate human intervention by a professional.

However, machine learning algorithms operate differently. A program is fundamentally trained using the data it receives and attempts to discover patterns between occurrences. In successive executions, the software consults the data-driven model to change the input and generate a result (while updating its data model). Thus, such algorithms become dynamic and adaptive to various forms of input without the need for human interaction. Additionally, the program's capacity to cope with a variety of scenarios is not constrained by the programmer's skills.

Intrusion detection systems utilize two ways to identify intrusions: misuse detection and anomaly detection. Misuse detection identifies anomalous states (for example, malicious packets) and attempts to locate them within the data set. Traditional Intrusion Detection Systems do this by examining the database of prior attacks and comparing them to the packets coming on the network (i.e., the headers and contents of packets on the network).

Detecting network abnormalities is a different story. In this case, the system must recognize the pattern in the data, classify a condition as 'normal,' and then treat any deviation from that definition as a possible threat. This cannot be accomplished using conventional Intrusion Detection Systems, as previously stated, their skill is matched by their programmer's.

A threat is unpredictable by definition, which is what makes it a threat in the first place. Machine learning techniques can control and identify behavioral problems, which is why there is a growing trend of employing Artificial Intelligence (AI) software in conjunction with IDS[4]. The survey conducted of Hongyu Liu and Bo Lang[5] presented a list of related implementations and methodologies. Some of these will be shown in the following chapters.

The most frequently encountered issue in machine learning research is the collection and structuring of data. As machine learning-based intrusion detection systems are a relatively new technology, it's logical that not as much work has been done on developing datasets to "train" our algorithms. The majority of dataset generating implementations employ Packet Flow data (comparable to Cisco's NetFlow[6] data) and utilize metrics such as packet delivery rate, connection counts, and so on to identify network hazards. While this is an effective method for detecting the most damaging sorts of threats (e.g., brute force assaults), it is ineffective at detecting more complicated threats in which just a few fields of packets are modified to carry out the attack (for example the Ping of Death attack). Certain implementations augment their data with protocol header fields such as the TCP FLAG. However, the header fields included in those solutions were inserted because of previous knowledge indicating that an attacker may utilize this flag to launch an assault (e.g. SYN Flood). As a result, if a new zero-day assault exploits a vulnerability in another field (for example, TCP Options), the Intrusion Detection System will be unable to detect it.

The Al Pastor project was implemented during this thesis. The project's goal is to create datasets from local networks, analyze network traffic, and detect risks using Neural Networks. The system generates two kinds of datasets: traditional Netflow data and protocol-specific data. We employ a rule-based intrusion detection system to assign a value (1-4) to suspicious packets depending on their riskiness. We next train various Neural Network models, that we refer to as Experts, to classify incoming packets (and packet flows) based on Snorts' classification.

There are Protocol Experts in addition to Netflow Experts. We can examine and decide if new packets coming in our network are risky or not once we train the models using datasets we produce (or datasets acquired from other sources in the pcap file format). Our experiment shows that when learning approaches are taught with a mix of Netflow data and packet headers, we may get superior outcomes in intrusion detection.

The chapters that follow are organized in the manner outlined below:

- The first chapter discusses network assaults, their classification, and instances of such attacks.
- In Chapter 2, you'll learn about intrusion detection systems and packet analysis systems.
- Machine learning and neural network theory are introduced in Chapter 3.
- In Chapter 4, we will look at intrusion detection systems that make considerable use of machine learning techniques and freely available datasets.
- Chapter 5 describes the structure of the Al Pastor program as well as the results of the tests we conducted.

Chapter 1

Network Attacks

1.1 The consequences of a network assault

The usage of computers is no longer optional in today's world; it is required. A few years ago, a screen separated cyberspace from us. It was a reminder that our operating system, software, and internet are all realms contained within our computer box. However, the convergence of machine and man (and, to some extent, their union) results in an unsettling symmetry: Because the human world has gotten closer to the software world, software risks have become threats to humans.

The Internet of Things is a prime example of this kind of fusion of man and machine. The "real" world, the world of humans, is now populated by physical devices that run software, are networked, and are capable of communicating with one another.. Some execute simple procedural algorithms, while others make decisions using the power of Artificial Intelligence. Because all of those Things are linked to the Internet, they are vulnerable to hacking attempts and hence may be commandeered and used against us.

The danger was already understood even when ordinary computers were used. A malicious person who obtains credit card information or access to another person's private messages has the potential to have a significant impact on their personal lives. Consequently, a hacker can indeed utilize software to inflict damage (in the "real" world) by manipulating personal data. Such assaults, however, do not represent a direct danger to our physical survival. An attacker, on the other hand, may wreak havoc and pose a genuine physical danger to our lives by accessing equipment in our surroundings (for example, a stove or a garage door) or exploiting and manipulating our cars (Internet of Vehicles - IoV).

However, the danger of cyber-attacks may grow more widespread. New terminology, such as "cyberterrorism"[7] and "cyberwarfare"[8] are being incorporated into our vocabulary to describe the use of digital assaults against whole nations. The assault on Esthonia in May

2007[9] was one of the earliest instances of a cyber war. Unknown attackers attacked banks, media outlets, and government organizations, during this event, by deleting their websites. Since then, cyber warfare techniques have been added to the list of tools of states around the world. In 2019, America retaliated to tanker strikes, for which it held Iran responsible, with a cyber assault aimed at seizing control of military systems[10]. Cyberwarfare, in general, is not a novel concept. There have previously been instances of one state orchestrating cyber assaults against another (for example, Stuxnet's use against Iran's nuclear plants[11]), but these would not be considered acts of war.

As a result, we infer that cyber assaults can be directed at gadgets in our homes, bank accounts, corporate websites, or even entire countries. In theory, no one is secure as long as they are connected to the internet. Now, more than ever, we want dynamic security against cyber-attacks, capable of defending against known dangers while identifying new ones[4].

1.2 Taxonomy of attacks

Attack classification has always been a significant technique for security specialists to better identify possible attacks on a given information system. By grouping assaults, one may swiftly concentrate on particular forms of attacks instead of having a big list of assaults that would take more time to study and choose attacks of interest.

W. Stallings gives a list of the major security services in his book "Cryptography and Network Security: Principles and Practice" [12], i.e. "services provided by a protocol layer of communicating open systems that ensures adequate system security or data transfer security". The most important security services are:

- *Confidentiality*: Confidentiality refers to the protection of data, objects, and resources against unwanted sight and access.
- *Authentication*: Authentication is a service that is used to establish the identity of the person who generated the information. It makes sure that hosts, really represent that which is said to be represented.
- *Integrity*: Integrity safeguards guard against unwanted data change. These controls certify the correctness and completeness of data.
- *Availability*: Data availability refers to the ability of users to access data. It ensures that users may access your system and data whenever they are required. It is frequently related with system stability and uptime.

The types of attacks that are capable of compromising each of the aforementioned security services is a typical attack classification[12].

- *Interception (Confidentiality)*: Interception attacks encompass all types of assaults aiming at jeopardizing the secrecy of critical infrastructure.
- *Fabrication (Authentication)*: Fabrication attacks involve the use of a system to generate data, activities, interactions, or other activities with the intent of simulating the behavior of another system. Although fabrication assaults are primarily concerned with integrity, they can also be viewed as a danger to availability.
- *Modification (Integrity)*: Modification attacks include a variety of different types of attacks aiming at altering the content of messages and data storage.
- *Interruption (Availability)*: Interruption attacks are directed at depriving lawful parties of access to a set of infrastructure services.

Those attacks can be executed by exploiting vulnerabilities. A vulnerability or flaw is a weakness in the security design of a system that may be exploited by attackers. The overall vulnerability of the system to threats is determined by the likelihood of an attacker exploiting those faults.

1.3 A brief overview of the TCP/IP Protocol Model

Before we present prominent attacks and attempt to classify them, it is necessary to define and demonstrate the channel via which network attacks are transfered from one machine to another. The internet is this medium, and the TCP/IP model is what binds it all together.

The Internet Protocol Suite (TCP/IP) model was created to define the functionality of a communication system by breaking the communication method down into smaller, simpler components (a nice compact overview is provided here [13]). It enables us to decide the best way for a particular computer to connect to the internet and how data should be delivered between it and other computers. It was established in the 1960s by the United States Department of Defense (DoD) and is based on industry standards. The Internet protocol suite makes no assumptions about the hardware or software environment in which it runs. It is sufficient to have hardware and software capable of transmitting and receiving packets across a computer network. As a result, the suite is now available on virtually every computing platform. This section will examine the TCP/IP model's format and numerous protocols. Additionally, we will investigate a list of threats and attacks to this technology stack and attempt to categorize them.

1.3.1 Structure

In general the TCP/IP Protocol Stack is defined by its three principles:

- *End-to-End*: A network design approach which preserves application-specific functionality at communication end points. It essentially removes crucial components from intermediary communicating nodes that serve only as data conduits.
- *Robustness*: Otherwise known as Postel's Law, which is best articulated by his own phrase, "Be liberal in what you accept, and conservative in what you send."[14]. It essentially states that hosts should always deliver well-formed packets and accept any datagram that they are capable of decoding.
- *Encapsulation*: Encapsulation is the procedure of transferring data from one protocol to another via a translation process. This is performed by surrounding the data with headers and trailers (encapsulating them). Receivers should appropriately decapsulate the data in order to decode it.

The model incorporates these ideas throughout its four layers, which are described below:

- *Link*: The link layer is in charge of transferring data between hosts (or routers) over a single physical link. It serves as the foundation for computer communication. The internet is effectively made up of several such links between hosts. Ethernet (IEEE 802.3) is the most widely known Link Layer Protocol.
- *Internet*: This layer's primary role is to enable hosts to import packets from any part of the network and guarantee that they transit independently to their destination. Notable protocols are IP, ICMP, IGMP
- *Transport*: The transport layer is responsible for host-to-host communication, whether on a local network or a remote network divided by routers. The most commonly used Transport Layer protocols are TCP and UDP.
- *Application*: It takes into account the intricacies of data formatting and presentation. This layer makes use of the preceding layers' functionality to send messages to distant processes and to allow our programs to receive packets. HTTP, the internet's most famous protocol, belongs to this layer.

Data is transmitted between hosts using the TCP/IP principles and layers. When data is sent from the Application layer to the TCP/IP model's lower layers at the transmitting machine, layer protocols include some sort of header information and the Link layer includes

a trailer (Encapsulation). Finally, the data is transmitted through a network cable as a stream of bits. When data is received at the receiving computer, each layer removes the header information and uses it to transfer the data to the next higher layer or to reassemble the data (Decapsulation). Finally, the data sent from the transmitting computer is handed up to the receiving computer's application layer.

From here on, when we refer to a packet as $proto_1/proto_2/.../proto_n$, we mean a packet having several layers of encapsulation, with the first packet header being of type $proto_1$, the second being of type $proto_2$, and so on. We will be later using this notation to describe Protocol-Specific Neural Network Models and Datasets.

1.3.2 TCP/IP Protocols

We will briefly cover the most critical protocols, and their headers, in the TCP/IP stack.

Ethernet (Link Layer)

Ethernet specifies the format of the data at the data connection layer, including the header and trailer. The protocol is established in the venerable IEEE 802.3 standard. It uses MAC Addresses to uniquely identify communicating hosts.

ICMP (Internet Layer)

The Internet Control Message Protocol (ICMP) is used in order to forward error messages and control information across different hosts in the network.

IPv4/IPv6 (Internet Layer)

A collection of rules that govern the manner in which data should be transmitted across a public network (Internet). Frequently, it is used in combination with the transmission control protocol (TCP). The IP Addressing scheme is a part of the IP protocol. By using IP Addresses, hosts are able to refer to themselves or other hosts of the network. IPv6 is a newer more sophisticated version of IPv4, yet both are used interchangeably to maintain backward compatibility.

ARP (Link Layer)

ARP supports the IP protocols by associating the IP addresses of various hosts in our network with their MAC Address. When connecting with one another, Arp Clients do inquiries

("Who owns the *X* IP address") to determine where to send packets (to which MAC Address destination).

UDP/TCP (Transport Layer)

Both enable apps or processes to communicate with one another and introduce the concept of a port (a unique identifier for different processes running on a host machine). Packets are delivered to the relevant process by means of the port. The two protocols are functionally distinct: TCP is more trustworthy (it does not drop packets), provides congestion control, and establishes a connection before any real communication takes place, whereas UDP is connection-less, less reliable but significantly quicker.

SSL/TLS (Transport Layer)

Secure Sockets Layer (SSL) is a commonly used security technology that enables privacy and data confidentiality for Internet interactions. TLS (Transport Layer Security) is an improved, more secure version of SSL. SSL/TLS runs on top of the TCP Protocol.

DNS (Application Layer)

The Domain Name System (DNS) makes it easier for the general public to explore the Internet by linking familiar domain names to the numeric network addresses needed to send data over it.

HTTP (Application Layer)

The Hypertext Transfer Protocol (HTTP) is a TCP-based protocol for transmitting hypermedia content from a server to a client. When we access a website via the browser (client), HTTP is utilized to retrieve the website's data from a remote server. The protocol is composed of two sorts of packets: a request and a response. When a request is sent, the TCP connection between the server and the client is maintained until the complete HTTP response is sent.

1.4 Attacks against the TCP/IP Model

Since we've already discussed the TCP/IP concept, we'll now go into numerous forms of attacks on the protocols themselves.

1.4.1 DoS Attacks

A Denial-of-Service (DoS) attacks attempt to bring a computer or network to a halt, rendering it unreachable to its users. They can thus be classified as Interruption attacks. Fifteen year old Hacker "mafiaboy" is documented as the first person ever to instigate a DoS attack [15]. Initially, he targeted academic websites before focusing on bringing down big websites using his tools. He claimed he did it to get attention in the hacking community. Since then more and more types of DoS attacks are added into the hackers' arsenal, we'll go through some of the more prevalent ones here.

The Ping Of Death

A vulnerability in some target systems allows for Ping of Death (PoD) attacks[16]. IP payloads above 65KB might result in a memory overflow in early TCP/IP implementations, which were unable to manage such a large payload. When a victim receives numerous similar packets, the assaults are carried out in the same way. Despite the fact that newer systems have been patched to address these kinds of security issues, the attacks remain important since vulnerable hosts still exist.

SYN Flood

The TCP protocol is establishing a connection before attempting to send any data packets. The connection establishment is achieved by implementing the TCP SYN-ACK handshake. The process is described bellow:

1. Client sends a SYN message to the Server
2. Server responds with a SYN-ACK packet and stores client information locally
3. Client responds with an ACK packet and thus the handshake is complete

To carry out the attack[17], the attacker takes advantage of the handshake's second phase. They begin by sending SYN packets to the server. which stores new client data for each new SYN packet, and then ignores the servers' SYN-ACK response. Thus, the server continues to store new client information in its memory until there is insufficient memory available to serve legitimate users.

Slowloris

Slowloris[18] exploits HTTP in the same manner as SYN Floods take advantage of the TCP protocol. It communicates with a server using HTTP Requests and then keeps the connections

open rather than terminating them. This is accomplished by sending incomplete packets that the server successfully processes (robustness) and is fooled into believing the remainder of the packets will follow. At some point, the server will run out of connections and will refuse to serve additional customers. Slowloris is distinct from other denial-of-service attacks in that it transmits packets at a considerably slower rate.

1.4.2 Malware

Malware (short for malicious software) is a colloquial phrase that refers to a wide variety of destructive or intrusive software types, including the following:

- *Computer Viruses* Software that spreads between different hosts. Their activation can only be triggered by their host.
- *Worms* Which are very similar to Computer Viruses. However their activation can be initiated independently of their host.
- *Ransomware* The attack consists of two steps. First the attacker encrypts the files on the victim's device, and then it demands a ransom in order to decrypt it.
- *Trojans* Malware that tries to avoid detection by masquerading as a benign packet (or program, etc.).

It can be executable code, scripts, active content, or other applications. It is therefore usually transmitted via the data section of application (or transport) layer protocols. In addition to stealing, decrypting, or erasing critical data, these Malware can also monitor users' computer activities and spy on them. Malware attacks can be classified as Modification and Interception type. However, certain malware may also be classified as Fabrication since they can obtain user data and then use it to impersonate the user.

1.4.3 Design/Implementation Exploits

The following attacks were only feasible as a result of poor protocol design or implementation.

Heartbleed

An attacker might obtain access to memory data via a vulnerability in the OpenSSL library (which implements the SSL/TLS protocols). The attack was made feasible by abusing the SSL/TLS protocol's Heartbeat feature[19], which enables connections to remain active for

an extended period of time without requiring fresh connection requests[20]. The Heartbleed TLS extension works as follows:

1. A client sends a message to the server requesting to replay the contents of the message. The packets also includes the length of the message.

2. The server responds back with the given text

However, the OpenSSL Heartbeat implementation requested exactly the number of bytes specified in the heartblead's payload (Figures 1.1 and 1.2). This implies that if the client's

```
/****************************/
/** snippet from n2s macro **/
/****************************/
/*Payload can be up to 64KB, defined by user*/
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;
...
/* Moving payload size to bp, could cause overflow*/
memcpy(bp, pl, payload);
```

Figure 1.1 Heartbleed: Dangerous memcpy call

```
hbtype = *p++;
/*n2s contains an unsafe memcpy call*/
/*operation which allows memory leakage*/
n2s(p, payload);
pl = p;

```

Figure 1.2 Heartbleed: Calling n2s without checking payload length

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
  return 0; /* silently discard */
hbtype = *p++;
/* safe memcpy operation */
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
  return 0; /* silently discard per RFC 6520 sec. 4 */
pl = p;

```

Figure 1.3 Heartbleed: Vulnerability Fix

real text is less than the given quantity, the server will attempt to transfer more bytes which it would get from its own memory, thus transmitting sensitive data to the attacker. A later

fix was made shortly after the Attack was published (Fig. 1.3), however a large number of attackers exploited it in order to get access to personal data.

Downgrade Attacks

The TLS protocol supports backward compatibility by allowing clients or servers to downgrade to an earlier SSL/TLS version during the negotiation process[21]. An attacker can conduct a Man in the Middle attack (by inserting himself between a client and the server) and then send bogus downgrade requests to the clients. These downgrade requests may result in the two communicating parties utilizing older versions of protocols known to be vulnerable to a variety of vulnerabilities (for example POODLE for SSLv3).

FREAK Vulnerability

An additional TLS vulnerability[22]. It is based on some legacy OpenSSL capability, the EXPORT Ciphersuites. These ciphersuites enabled users to utilize smaller cryptographic keys (512 bits) with the RSA ciphersuite. Attackers can factor the RSA modulus of 512 bit integers reasonably easily on current computer systems. An attacker may easily use a MiTM attack to insert himself between the client and server, alter the initial client TLS message (CLIENT HELLO) to request EXPORT RSA security, and then factor the server's 512 bit key. Finally, the attacker can get the Master Key, which is utilized for all further communication.

1.4.4 Tracing such attacks

It is obvious that these assaults have several traits, which may be utilized to track them down even if we had no prior knowledge of them. For example, a DoS assault may be identified by the large amount of packets traveling over the network, or by the unusually high number of open connections, as is the case with Slowloris. However, attacks such as the Ping Of Death can also be recognized by inspecting header data (payload size > 65K).

On the other side, malware may be identified by looking for programs or code embedded in the payload of the packets. As a result, techniques such as malware fingerprinting are employed. However, they are not always simple to identify since hackers may modify the message's structure or wording (for example by using escape characters).

Finally, by examining the headers of the packet, attacks such as Heartbleed, Downgrade Attacks, and Freak may be recognized. Freak may be tracked if, for instance, a software recognized that employing 512 cipher keys in conjunction with TLSv2.0 is exceedingly rare. Similarly, a software may have detected a Heartbleed attack by noticing that the length of a

TLS Heartbeat message differed from the length stated on the packet. As a result, packet header analysis is required to detect and characterize such irregularities in packet data.

The next chapters will explore how combining machine learning and classical IDS may be used to trace both known and unknown threats using packet header and traffic analysis.

Chapter 2

Intrusion Detection Systems

An IDS is an acronym for "intrusion detection system". An intrusion is defined as unauthorized access to information within a system that jeopardizes its credibility, security, or performance. On the other side, the detection system is a defense mechanism for identifying such unlawful behavior[23][24]. In general an IDS is a monitoring tool that continuously analyzes host data for any behavior that violates the security policy and jeopardizes its confidentiality, integrity, and availability.

While IDS are meant to passively monitor traffic and raise warnings when suspicious traffic is discovered, Intrusion Prevention Systems are designed to actively attempt to prevent the attack from succeeding. Intrusion Prevention Systems (IPS)[25] are a novel approach to network defense that effectively combine the firewall and intrusion detection techniques. This proactive approach prevents attacks from entering the network by examining various data records and the detection behavior of pattern recognition sensors. When an attack is identified, intrusion prevention first attempts to take action (block the incoming malicious data) and then it logs the respective alerts.

This thesis will be mostly concerned with identifying threats (IDS). The system's response to discovered threats will not be discussed here.

IDS can be classified based on the manner of its deployment or detection methods[26]. We will be reviewing those classifications in the next sections.

2.1 Deployment method based IDS

Based on whether the IDS are deployed on the hosts of the network itself we can distinguish the into two categories: a) Host-based intrusion detection system (HIDS) and b) Network-based intrusion detection system (NIDS).

2.1.1 Network-based Intrusion Detections Systems

NIDS are intelligently distributed sensors that passively monitor traffic passing through the devices on which they are installed. NIDS can be hardware devices or software programs that connect to a variety of network media depending on the vendor. NIDS frequently features one network interface for promiscuous listening to network talks and another for control and reporting [27].

2.1.2 Host-based Intrusion Detections Systems

On the other hand, HIDS examines system behaviour for signals of odd activity and operates on the host. To identify changes, Host-based IDSs often monitor running processes, and memory usage (as well as changes in the filesystem such ass permission edits, file creations, etc). By itself, a host-based IDS is not a full solution. Although it is rational to monitor the host, there are some drawbacks:

- It influences the host's performance
- Attacks can be detected only once they have affected the host
- It must be installed on all hosts that require intrusion prevention and detection.

In order to combine their strengths, host-based and network-based IDS are frequently employed simultaneously[28].

2.2 Detection Methods

HIDS and NIDS can both utilize signature-based or anomaly-based detection methods, depending on the threat. For a more complete approach, some IDS solutions can even combine the two detection approaches.

2.2.1 Signature Based

Signature-based intrusion detection systems (IDS) are designed to identify and block attacks based on the patterns and sequences of traffic that arrive on a network such as network headers and data that match known malware. In general, this form of detection detects malicious patterns by looking for signatures in a given set of indicators (Indicators of Compromise - IOCs[29]) such as:

- Fluctuating amounts of requests and reads in the company's data Network traffic that goes through seldom utilized ports
- Suspicious use of administrator or privileged account passwords
- Unusual traffic with countries with whom no packet exchange has occurred previously
- The network is experiencing abnormally high traffic levels.
- Probing or brute-force attacks are indicated by strange log-ins and network access.
- On the system, there are unidentified files, applications, and processes.

In general, the inability of signature-based IDS systems to identify unknown attacks is a significant disadvantage. In order to escape detection, malicious actors might simply alter their attack sequences in malware and other sorts of assaults. Furthermore, those systems, are completely ineffective when dealing with new zero-day attacks.

2.2.2 Anomaly Detection

The "behavior-based IDS" hypothesis of Anomaly IDS[30](AIDS) proposes to accurately define a normal life profile. An anomaly is defined as a deviation from this standard profile. It is possible to predetermine acceptable behaviour or to learn about it through the requirements and criteria that the system administrator defines. There are two stages to the development of Anomaly-based IDSs: training and testing. A model of typical behavior is first learned from the usual traffic profile, and then a fresh data set is utilized in the testing phase to check if the system can generalize to previously unknown incursions. There are three major techniques when designing AIDS[31]:

- *Statistics-based techniques*: This technique makes an attempt to identify dangers based on the statistical chance of an event occurring. Threats are defined as events with a low likelihood of occurring.
- *Knowledge-based techniques*: This technique demands the establishment of a knowledge base that correctly reflects the genuine nature of events, in which deviations from the norm are deemed incursions. Knowledge-based techniques are also known as the Expert-System Method.
- *AIDS based on machine learning techniques*: Anomaly-based IDS systems have been developed using machine learning techniques. Using these methods, the amount of human intervention required is maintained to a minimum.

2.3 Metrics

Numerous indicators have been developed to assess IDS's performance. Most of them are based on the Confusion Matrix (see Table 2.1), a two-dimensional matrix which depicts the classification results for the overall dataset. Those results are composed of the following metrics:

- *True Positives (TP)*: The classifier correctly classified the data occurrences as Attacks.
- *False Negative (FN)*: It was assumed mistakenly that the data instances were Normal.
- *False Positive (FP)*: Examples of data were wrongly identified as an assault.
- *True Negative (TN)*: The incidences were classified appropriately as Normal.

Confusion matrices aren't particularly beneficial when it comes to comparing IDSs. The confusion matrix variables, however, are used to construct different performance indicators[32].

- *Precision (PR):* It is the percentage of predicted Attacks that are correctly anticipated to be Attacks.
$$PR = \frac{TP}{TP+FP}$$
- *Recall:* It is the percentage of samples classified properly as Attacks to all samples classified as Attacks. Additionally, the term Detection Rate (DR) is used.
$$Recall = DR = \frac{TP}{TP+FN}$$
- *False alarm rate (FAR):* It is referred to as the false positive rate. It is defined as the ratio of wrongly predicted Attack samples to all Normal samples.
$$FAR = \frac{FP}{FP+TN}$$
- *True negative rate (TNR):* It is defined as the percentage of correctly classified Normal samples in comparison to all Normal samples.
$$TNR = \frac{TN}{TN+FP}$$
- *Accuracy :* It is the ratio of properly classified events to the total number of occurrences. It is sometimes referred to as Accuracy of Detection or Classification Rate and is only

useful as a performance statistic when a dataset is balanced.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

- *F-Score*: It is a metric for determining the reliability of a test.

$$FM = \frac{2}{\frac{1}{PR}+\frac{1}{DR}}$$

Table 2.1 IDS Confusion Matrix

		Predicted	
Actual		**Attack**	**Normal**
	Attack	True Positive (TP)	False Negative (TP)
	Normal	False Positive (TP)	True Negative (TP)

According to the bibliography, 22 measures are generally used to evaluate IDS empirically. With so many indicators available, evaluating the performance of different IDS is difficult, as authors analyze each IDS differently. Very few, if any, studies have attempted to construct a de facto metric system for comparing IDS techniques[32][33]. We will use the metrics mentioned in this part to evaluate the performance of the IDS we created in this thesis.

Chapter 3

Machine and Deep Learning

As currently defined, machine learning is a field of research that focuses on the use of data and algorithms to simulate how humans learn, gradually improving its accuracy.

The technique is fundamentally based on two key concepts:

- *Self-Learning*: The process of analyzing data and identifying patterns through the use of algorithms. Unlike conventional programming, which relies on explicit rules and decision trees defined by the programmer to reach a conclusion, machine learning relies on data to develop a decision model. Machine learning makes use of complex algorithms, trial and error, probabilistic reasoning, and other techniques. Thus, the decision model is formed based on the substance of the incoming data, rather than on pre-defined human-programmed criteria.

- *Improvement through Experience*: Machine learning improves its predictions through data exposure by evaluating the success or failure of past attempts. As a result, machine learning accuracy is highly dependent on data in order to modify and enhance the model's weak assumptions.

3.1 Concepts

By analyzing the success or failure of prior attempts, machine learning improves its predictions. As a result, machine learning accuracy is highly dependent on data. The following are the most prevalent challenges that we are attempting to answer with machine learning:

- *Regression*: Attempting to guess the output variable when it is a real or continuous value. For example, attempting to predict the price of a certain stock based on the performance of comparable stocks in the past.

- *Classification*: Problems where the output is a category. In such problems the point is to "categorize" the data. For instance, classifying a network packet as a threat or as benign.

This section will look at aspects machine learning, specifically the datasets, feature selection, and training phase. In general, datasets are made up of rows and columns. The features are the accessible columns (for example, "temperature", "date", and so on), whereas the label is the value that the model is attempting to predict (for example, tomorrow's stock price, the "niceness" of a network packet, and so on).

3.1.1 Dataset

The word "Dataset" refers to the process of collecting or producing the data that will be utilized as input to the model during training. Data can either be presented in a structured or an unstructured form. Structured data is data that has been organized, defined, and labeled and can be arranged in rows and columns, whereas unstructured data lacks any organizational structure that may be arranged in a table. Numbers, dates, booleans, and strings are all instances of structured data whereas songs, videos, images are all instances of unstructured data. Unstructured data, as predicted, demands far more storage space than structured data. Data gathering is a critical phase in machine learning, as erroneous, skewed, or inaccurate data might result in undesirable or unexpected outcomes.

Data can be obtained or developed in-house or derived from publicly available datasets. In any case, raw data may be incomplete, include mistakes, or be missing portions. As a result, pre-processing of data is required before it can be used by the machine learning algorithm. Data cleaning, data imputations, oversampling, data integration, and data normalisation are all examples of pre-processing techniques. Data is then spit into three segments:

- *The Training dataset*: the dataset used to fit the model's parameters.
- *The Validation dataset*: A subset of data used to modify the model's hyperparameters while also giving an unbiased assessment of the model's fit to the training data. Validation datasets can be used to improve the model's regularisation. When the error rate of the validation data increases, training is halted, as this indicates that the model has been over-fit to the training dataset.
- *The Test dataset*: It is utilized only after the model has been fully trained to evaluate the final model. To do this, the final (trained) model classifies instances from the test set. These predictions are compared to the real classifications of the cases to determine the model's accuracy.

3.1.2 Feature engineering

It's possible that the raw data is incomplete or noisy. To be of any value to machine learning, raw data must be processed and cleansed before usage. Feature Engineering is a critical step in machine learning since it helps minimize data dimensionality and isolate the most critical and useful aspects of the dataset, all while simultaneously reducing computing burden. The practice of feature selection is used in feature engineering to exclude from the dataset any characteristics that would increase computation at no practical benefit. To further simplify data processing, the feature extraction technique merges some of the features together into new ones.

3.1.3 Data Preprocessing

After picking a set of features for training the models, the following step is to convert and parse the feature values so that the machine can readily understand them. Some commonly used preprocessing techniques are presented bellow.

One-Hot Encoding

It is rather usual for a dataset to have nominal categorical values. One-Hot Encoding (OHE) is a frequently used approach for such values. OHE divides the original feature into as many features as the nominal variable's maximum number of potential values . Instead of setting the value of the feature to its nominal value for each row of data, we place a one or a zero in the respective newly created column for each row. For example, let us consider a datasset where a feature is called "protocol_type" and it contains values such as "tcp", "udp", "icmp". By using OHE this feature is split into three distinct ones: "protocol_type_tcp", "protocol_type_udp", "protocol_type_icmp". A row where the value for "protocol_type" was set to "tcp" will now be transformed into a row where the values of "protocol_type_udp" and "protocol_type_icmp" are set to 0 and the value of "protocol_type_tcp" is set to 1.

Standardization

Standardization's ultimate purpose is to reduce all characteristics to a single scale without altering the range of values. Standardization reduces the mean value to zero and creates a distribution with a unit standard deviation. For every feature, its values are calculated using the following formula:

$$X' = \frac{X - \mu}{\sigma}$$

where μ is the features' mean value and σ their deviation.

Normalization

Normalization, like Standardization, attempts to rescale a collection of supplied values. However, it attempts to fit them into a range of values between 0 and 1. The equation applied to every value is the following

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

where X_{max} and X_{min} are the maximum and minimum values in the dataset for this feature respectively.

3.1.4 Learning

All machine learning tasks may be classified according to one of the following methodologies:

Supervised learning

Supervised learning is a technique in which a Model is given known input-output pairs and then attempts to analyze the relationship between the input and output data to learn about the underlying patterns linking the inputs and outputs. In order to develop the model, supervised learning requires a labeled dataset, with each object having labeled input and output values for model training. In order to categorize packets, the IDS shown in the last chapter employs Supervised learning.

Unsupervised learning

To produce new labels for probable outputs, Unsupervised Learning studies the relationships between input variables and identifies significant structure and patterns. Unsupervised Learning is a great tool to utilize when looking for relevant patterns and groupings in data, generating features, and doing exploratory work. Unsupervised learning is performed on unlabeled data.

Semi-supervised learning

A combination of unsupervised and supervised learning techniques that makes use of a big amount of unlabeled data in conjunction with a limited amount of labeled data. The model may be trained using labeled data and then used to categorize and discover trends for unlabeled cases.

Reinforcement learning

Highly similar to Ivan Pavlov's Classical Conditioning theory. The word 'conditioning' refers to the process through which a previously unconnected stimulus and reaction become correlated via learning. A fitness function is used in Reinforcement Learning to reward highly efficient agents (algorithms). The agents improve their efficiency by generating a result several times and then being or not being rewarded for it.

3.2 Machine Learning Algorithms

To demonstrate the application of the previously stated principles, we will examine two of the most extensively used machine learning algorithms, Naive Bayes and K-Nearest Neighbors.

3.2.1 Naive Bayes

Thomas Bayes, an 18th century mathematician and theologian, invented *Bayes' Theorem*, which was originally published in 1763[34]. It can be expressed as:

$$P(H|E,c) = \frac{P(H|c)P(E|H,c)}{P(E|c)} \tag{3.1}$$

which practically calculates the probability of an hypothesis H in light of new evidence E and prior setting c. The equation expresses the degree of belief in an assertion or propositions under the premise that other premises are true. The process of deducing attributes about a population or probability distribution from data is known as *inference*. To solve classification queries, the Naive Bayes' Classifier turns a given dataset into frequency tables and determines the posterior probability (training phase). Because the classifier presupposes that all features are unconnected to one another (thus the term "naive"), it is unable of comprehending or learning about the linkages and connections that exist between the features.

3.2.2 K-Nearest Algorithms

Another widely popular categorization approach is k-Nearest Neighbors (kNN). This is a technique for identifying new data using supervised learning based on its similarity to previously identified data. Although kNN is mostly used for classification, it may also be used for regression. Each row of data is treated as a distinct point in a multidimensional field by the algorithm. It then attempts to connect any new points with the class (or label) of the nearby points.

3.3 Deep Learning

Deep Learning is regarded as the evolutionary stage of machine learning. A deep learning model is supposed to examine data indefinitely using a logical framework similar to how a human would by using a technique referred to as Neural Networks (NN).

3.3.1 Neurons

We shall cover the notion of a Neuron before attempting to describe Neural Networks. A neuron produces a single output y_1 from multiple inputs $x_1, x_2, \ldots, x_n$ (see Fig. 3.1).

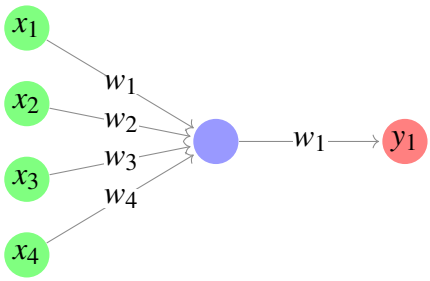


Figure 3.1 Neuron example

Each input has a weight attached with it $w_1, w_2, ..., w_n$. Each weight is a number indicating the relevance of the input to the output. When given an input the Neuron may or may not fire a value based on the activation function of the neuron. In general the neurons output may be defined as $\alpha = f(w \cdot x + b)$ where α is the output, f the activation function, w and x are the weight and input vectors and therefore $w \cdot x = \sum_j w_j x_j$. Finally, the negative threshold value or b is the neurons' bias.

A neuron is quite similar to how a neuron brain cell works. Whenever the stimuli are strong enough, the neuron outputs 1 similarly to how the neuron cell fires through its axon based on its dendrites input. For instance, let's consider the activation function 3.2. If the weighted total of each input is larger than a set threshold, the output is 1 (the neuron "fires"), otherwise it is 0.

$$\begin{cases} 1, & \text{if } x \cdot w - b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

A neuron that determines whether a person would laugh has assigned the three inputs "eating popcorn", "viewing a movie", and "studying" weights of 0.2, 0.5, and 0.05, respectively, and the laughing threshold is set to 0.6. If this individual consumes popcorn while studying, they will not chuckle, as the perceptron will not fire $0.2 * 1 + 0.5 * 0 + 0.05 * 1 = 0.25 < 0.6$.

If, on the other hand, it decides to watch a movie rather than study, the weighted input will be more than the threshold, causing the person to chuckle. $0.2*1+0.5*1+0.05*0=0.7<0.6$.

It is evident that by increasing the number of inputs, altering the weights, raising or reducing the thresholds and using different activation functions we may mimic various decision-making models. However, a single Neuron can not fully handle more complex scenarios.

3.3.2 Neural Networks

A Neural Network (NN) is comprised of sets of neurons forming three different types of layers. The input layer is located to the left and the neurons included inside it are referred to as input neurons. The output layer, on the right, includes the output neurons. Finally, the buried layers are referred to as the hidden layers (See Figure 3.2). These networks are referred to as feed-forward neural networks due to the fact that the output of one layer is utilized as the input for the subsequent layer.

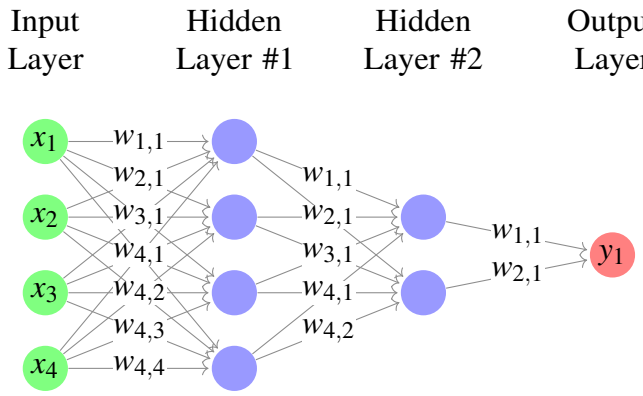


Figure 3.2 Neural Network example

The following is the general notion of training Neural Networks:

1. Assign values to input layer.
2. Observe the output layer's output.
3. Determine the error.
4. Adjust the weights and biases of each neuron.
5. Repeat

The error, or cost, is a metric indicating how "well" a neural network performed in relation to the training data and the predicted output. An example cost function is the Mean Square Error (MSE) presented on Eq. 3.3.

$$C(w,b) = \frac{1}{2n}\sum x||y(x) - \alpha||^2 \tag{3.3}$$

where vectors w, b are the weight and bias vectors while n is the number of inputs. $y(x)$ is the valid output for input x whereas α is our predicted output. As expected, MSE is minimized when $y(x)$ is close to α.

3.3.3 Activation Functions

There are several activation functions from which to choose depending on the sort of problem that the Neural Network is seeking to solve. The following are some of the most often used.

Binary Step

The Binary Step activation function (Fig. 3.3(a)) is threshold-dependent. When the neuron's estimated value reaches this threshold, the activation function sends a 1, otherwise it transmits a 0. This function is predominantly used for binary classification problems.

Sigmoid

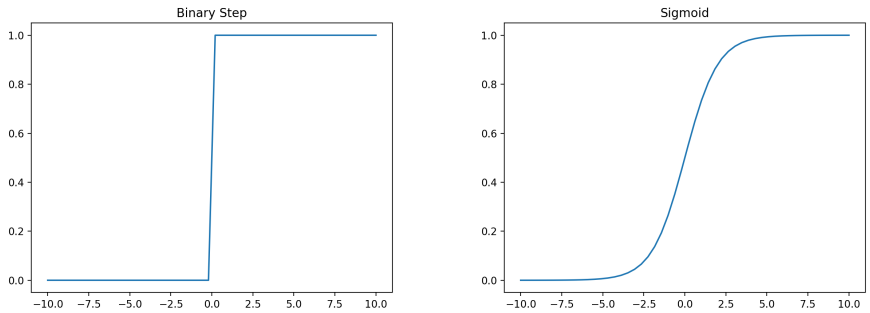
The Sigmoid function (Fig. 3.3(b)) normalizes the output of each Neuron. It is a subset of the logistic function and is often expressed as $\sigma(x) = \frac{1}{1+e^{-x}}$.

Rectified Linear Unit

The result of this function (Fig. 3.3(c)) can vary from 0 to infinity for positive inputs, however when the input is zero or a negative number, the function returns zero. It is a frequently used activation function that is mostly used to address classification problems.

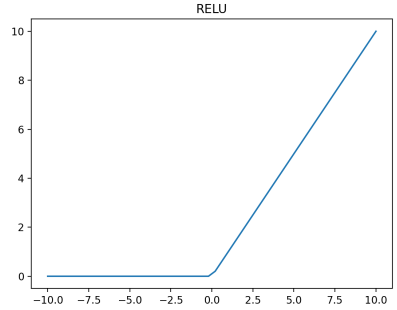
SoftMax

Softmax (Fig. 3.3(d)) is an extension of logistic regression that may be used for multi-class classification. Its formula is remarkably similar to that of logistic regression's sigmoid function.

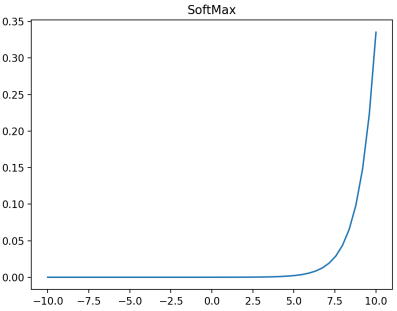


(a)

(b)



(c)



(d)

Figure 3.3 Activation Functions: (a) Binary Step (b) Sigmoid (c) RELU (d) SoftMax

3.3.4 Training

The primary goal of training a neural network is to decrease the error metric by modifying the weight and bias components appropriately.

Introduction to Gradient Descent

Gradient descent (GD) is an Optimization algorithm which attempts to trace local minimum/-maximum by iterating over some steps. A portion of the notation described below is based on Michael Nielsen's excellent introduction to Neural Networks blog piece[35].

Before attempting to discuss the mechanics of gradient descent with a large number of variables $(w_1, w_2, \ldots, w_n, b_1, b_2, \ldots, b_n)$, we will examine how to minimize the output of a simple three-dimensional function $z = 6x^2 + 2y^2$ (Fig. 3.4).

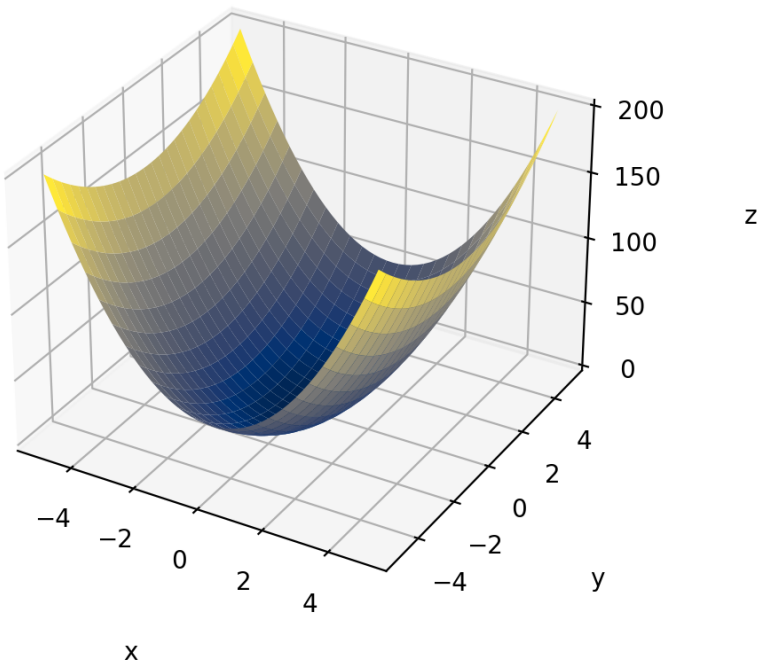


Figure 3.4 Function $z = 6x^2 + 2y^2$

When a small amount of movement $\Delta x, \Delta y$ is made in the x, y directions, the value of z changes as follows:

$$\Delta z = \frac{\partial z}{\partial x}\Delta x + \frac{\partial z}{\partial y}\Delta y \tag{3.4}$$

Given that we are seeking to reduce the value of z (the error), we must ensure that the direction change resulted in a negative value of Δz. As a result, we must devise a method for expressing the change in the z variable in terms of the vector of changes and z's gradient (the rate of change of the z function). Let us begin by determining the formulae for the final two:

- *Changes' vector*: $\Delta v = (\Delta x, \Delta y)^\intercal$
- *z's Gradient*: $\nabla z = (\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y})^\intercal$

Now that we defined Δv and ∇z we can finally express Eq. 3.4 as

$$\Delta z = \nabla z \cdot \Delta v \tag{3.5}$$

Since $||\nabla z||^2 \geq 0$, we can select values of vector changes so as

$$\Delta v = -\gamma \nabla z \tag{3.6}$$

Where $\gamma > 0$. γ, also called step or learning rate, determines the rate at which we will progress toward the ideal vector changes. Setting its value to a very large number may cause the local minimum to be skipped, whilst setting it to a tiny value may cause the minimum to take an inordinate amount of time to attain.

It is evident that by combining Eq. 3.5 and 3.6 we prove that produced Δz values will always be negative:

$$\Delta z = \nabla z \cdot \Delta v = -\gamma \nabla z \cdot \nabla z = -\gamma ||\nabla z||^2 \leq 0 \tag{3.7}$$

Finally, we may construct an iterative function that locates a local minimum given a collection of vector changes. Since $\Delta v = v' - v$ where v' the new position in the plane, we may write an iterative function as follows:

$$v' = v - \gamma \nabla z \tag{3.8}$$

Example

Given the example at Img. 3.4 ($z = 6x^2 + 2y^2$) and starting from a point A (10, 10) in the plot we may attempt to approach the local minimum by calculating the gradient:

$$\nabla z = (\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y})^\intercal = \begin{bmatrix} 12x \\ 4y \end{bmatrix}$$

.

If we set $\gamma = 0.1$ then

$$A' = \begin{bmatrix} 10 \\ 10 \end{bmatrix} - 0.1\nabla z(10,10)$$

$$A' = \begin{bmatrix} 10 \\ 10 \end{bmatrix} - 0.1 \begin{bmatrix} 120 \\ 40 \end{bmatrix}$$

$$A' = \begin{bmatrix} 10 \\ 10 \end{bmatrix} - \begin{bmatrix} 12 \\ 4 \end{bmatrix}$$

$$A' = \begin{bmatrix} -2 \\ 6 \end{bmatrix}$$

The value of z at point A was equal to $z(10,10) = 600 - 200 = 400$. At point A' the value of z is decreased $z(-2,6) = 96$. If we repeat the process one more time we are going to retrieve point A'' (0.4, 2.8) which produces $z(A'') = 16.64$

Using Gradient Descent to train DL Models

Due to the fact that Equation 3.8 operates on vectors of arbitrary length, it may be used to determine the local minimum of functions with more than two variables. For example, if z is a function of n variables $v_1, v_2, ..., v_n$ then:

$$\nabla z = (\frac{\partial z}{\partial v_1}, \frac{\partial z}{\partial v_2}, \dots \frac{\partial z}{\partial v_n})$$

.

Let's switch back to the original problem, calculating the minimum cost for a given set of weights w and biases b. We may determine the optimal weights for each neuron to neuron connection using the equation 3.8. The function that assigns a new weight for the connection i between two neurons in order to minimize the cost C can be expressed as:

$$w_i' = w_i - \gamma \frac{\partial C}{\partial w_i}$$

The same function can also be used to calculate the bias j of a neuron

$$b_j' = b_j - \gamma \frac{\partial C}{\partial b_j}$$

Numerous adaptations to the gradient descent approach have been developed; in this thesis, we will use the Adam Optimizer[36]. Adam is a widely used method in the field

of deep learning because it produces high-quality results quickly and outperforms a large number of other optimization techniques.

Backpropagation

We've examined how gradient descent works and how it can be used to train Neural Networks thus far, but we haven't explored how to compute the gradient of the cost ∇C with regards to every weight and bias value in the network. Backpropagation approach tries to calculate the needed list of C derivatives by starting at the output layer and gradually progressing backwards towards the hidden and input layers.

We will consider the output (activation) of the j_{th} neuron at the l_{th} layer to be expressed as $\alpha_j^l = f(z_j^l)$ and $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$. Also, w_{jk}^l is the weight of the link of the k_{th} layer of the $l-1$ layer to the j_{th} layer of the l layer, furthermore b_j^l is the bias of the neuron. It is evident how changes in z_j^l affect a_j^l and how the latter is influenced by the changes of the respective weights of the links between the previous layer and this neuron. Therefore the output of the neuron and consequently its error, are also affected by those factors. Thus, by using the chain rule, we can state that:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial z_j^l}{\partial w_{jk}^l} \frac{\partial \alpha_j^l}{\partial z_j^l} \frac{\partial C}{\partial \alpha_j^l} \tag{3.9}$$

and

$$\frac{\partial C}{\partial \alpha_k^{(l-1)}} = \sum_{j=0}^{n-1} \frac{\partial z_j^l}{\partial \alpha_k^{(l-1)}} \frac{\partial \alpha_j^l}{\partial z_j^l} \frac{\partial C}{\partial \alpha_j^l} \tag{3.10}$$

Calculation of the terms $\frac{\partial C}{\partial \alpha_j^l}$ and $\frac{\partial \alpha_j^l}{\partial z_j^l}$ is fairly easy and we can also determine that $\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1}$. Therefore we have everything we need in order to calculate $\frac{\partial C}{\partial w_{jk}^l}$. The steps of the backpropagation algorithms are the following:

1. Insert input data to the input layer
2. Retrieve the output vector from the output layer
3. Calculate the error C (via Cost Function)
4. Use Equation 3.9 to determine the derivatives of C with respect to the weights of all the links emanating from neurons in layer $l-1$ to layer l.

5. Use Equation 3.9 in combination with Equation 3.10, and repeat the computation for each one of the preceding layers until all components of the ∇C gradient have been determined

6. Repeat the same process in order to identify the gradient with regards to the biases (the bias derivatives can simply be calculated by exchanging the weight terms w^l_{jk} with those of the bias b^l_j in equations 3.9, 3.10)

Given that we now know ∇C, we may utilize it to create new weights and biases that attempt to lower the total error across a large number of iterations. This summarizes the neural network training procedure.

3.3.5 Recurrent Neural Networks

A recurrent neural network (RNN) is a type of artificial neural network that uses sequential or time series input to learn. Their distinguishing characteristic is their memory, which enables them to adjust the current input and output based on past data. These sorts of Neural Networks are generally employed to solve problems involving text, such as Natural Language Processing (NLP) and language translation.

RNNs compute gradients using the Backpropagation Through Time (BPTT) method[37], which is slightly different from classic backpropagation in that it is tailored for sequence data but follows the same concepts.

For instance, Image 3.5 depicts an RNN. On the left, we can see how the input x_i is relayed through hidden layer A, which outputs the value h_i and also provides feedback to itself. On the right of the illustration, the RNN is "unfolded" demonstrating how previous outputs influence future judgments about the value of h_i.

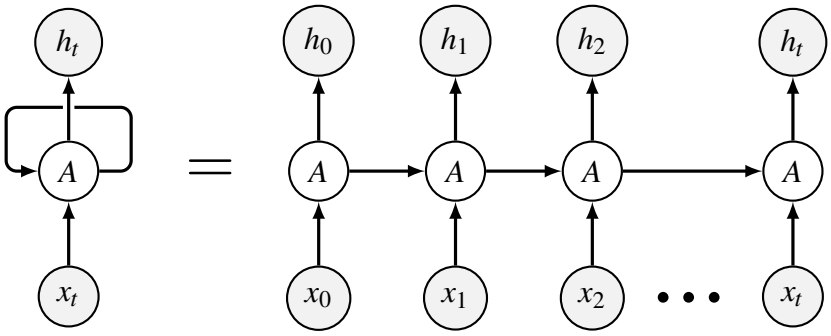


Figure 3.5 Unrolling an RNN through time

Chapter 4

Machine Learning and Network Intrusion Detection Systems

Network Intrusion Detection Systems (NIDSs) are critical tools for defending against increasingly complex cyber threats. Signature-based NIDSs (Section 2.2) compare attack signatures in order to identify a list of previously identified assaults. However, such systems are incapable of identifying previously unknown attacks or novel versions of existing attacks. NIDS can overcome the limits of signature-based IDS by using Machine and Deep Learning methodologies and eventually evolving into anomaly-based detection systems. We will be referring to such systems as Machine Learning - Network Intrusion Detection Systems (ML-NIDS). An attack on a network is begun by one or more malicious packets. As a result, the primary source of training data is labeled network traffic gathered from passive traffic listeners or generated artificially. The following steps (Fig. 4.1) summarize the fundamental methodology for training and detecting threats from intrusion detection systems:

1. Capturing internet traffic.
2. An existing threat detection system (signature-based) classifies packets depending on their association with an attack.
3. Some processing is performed on the packets' formatting (for example, Cisco's Netflow) (steps 2-3 can be done in a different order)
4. We choose the features of the packets to train the machine learning model on (Feature Selection).
5. The values of the features of the packets are preprocessed. As discussed previously, depending on how we manage post-processing data, we can do Standardization or

Normalization. Methods such as One-Hot-Encoding may also be employed in order to represent nominal categorical values.

6. The ML IDS is finally trained and its performance is evaluated.

7. From this point forward, ML-IDS either continues its training process (step 1) or directly assesses incoming network traffic

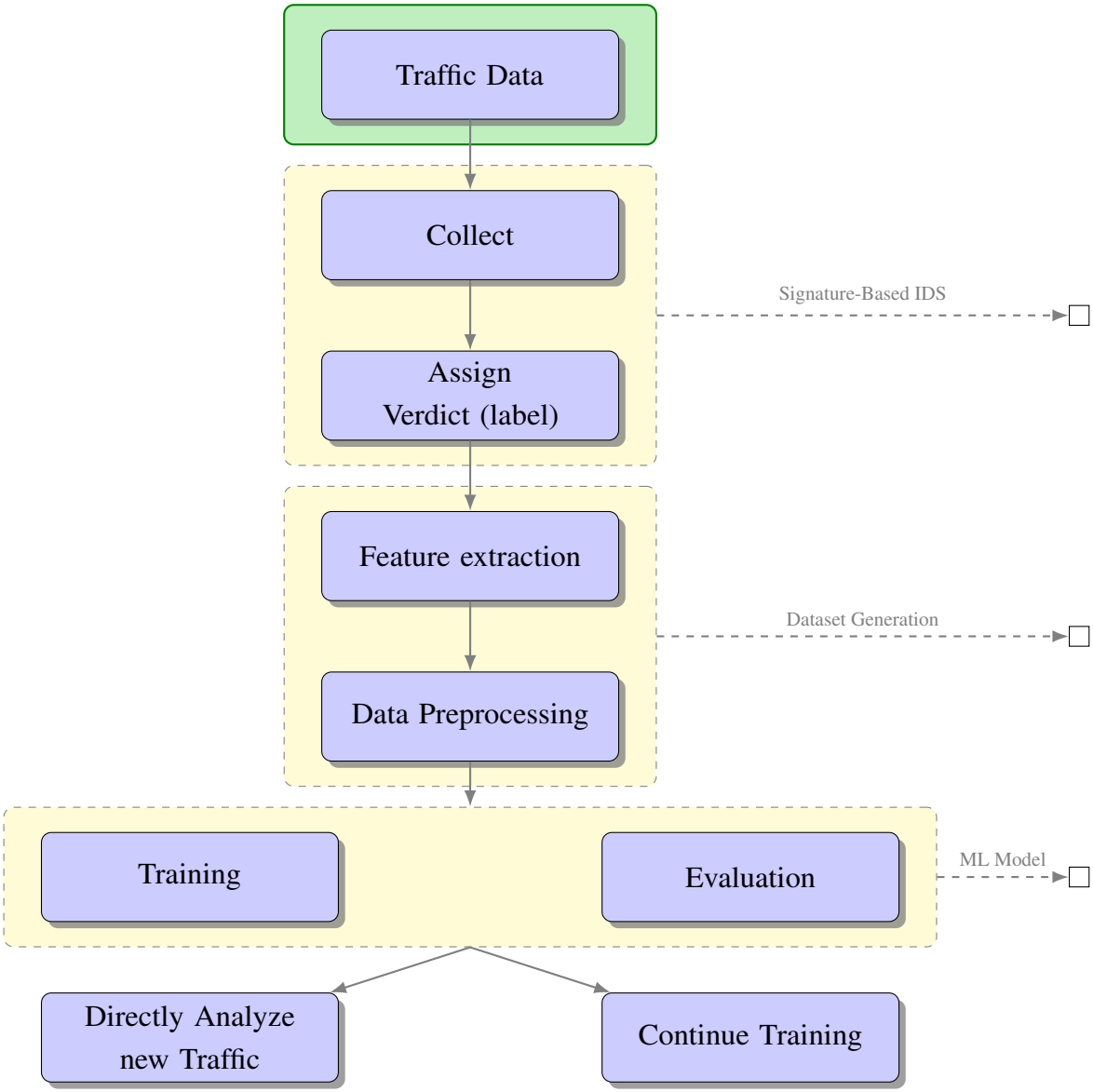


Figure 4.1 Training Process for ML-NIDS

Thankfully, new ML-NIDS do not need to go through the data collection and dataset development phases since they already have access to a pool of openly accessible datasets.

However, since those datasets are all unique, there is no practical method to compare either the datasets or the machine learning solutions based on them. Not only can datasets vary in terms of the sorts of data gathered (attack types, home vs office networks, normal vs artificial traffic etc), but also in terms of the features that their creators deemed relevant for training.

In the following sections we will explore some of the most well-known current datasets and also look at some unique techniques to integrating machine learning with NIDS.

4.1 Datasets

Typically, network packets are recorded in their original packet capture (pcap) format for creating datasets. Using proper tools and methodologies, a collection of network data characteristics is extracted from the pcap files, generating network data flows. As a consequence, a data source of labeled network flows illustrating benign and malignant network behavior is created. This section will cover the packets' flow format and several well-known and extensively utilized public NIDS datasets.

4.1.1 Packet Flow

By combining network packets into network flows, we may reduce the amount of data stored and also get a better understanding of the network's data traffic. Typically, packets are organized into flows using the five-tuple: protocol, source/destination addresses, and ports. Each flow is enhanced with additional data on traffic volume, statistics, and so on. Cisco have managed to establish their own network flow format, which has now become an industry standard: Netflow.[38].

Table 4.1 provides en excerpt from Netflow v9. Except for icmp and tcp flags, the majority of the information in Netflow is about the size and rate of bytes and packets transferred. To minimize confusion, we shall refer to network flows that diverge from the Netflow standards as emphPacket Flows.

Field	Length	Description
IPV4_SRC_ADDR	4	IPv4 Source Address
IPV4_DST_ADDR	4	IPv4 Destination Address
L4_SRC_PORT	2	Source Port (TCP/UDP)
L4_DST_PORT	2	Destination Port (TCP/UDP)
IN_BYTES	var	Number of incoming bytes
OUT_BYTES	var	Number of outgoing bytes
MIN_PKT_LNGTH	2	Minimum packet length
MAX_PKT_LNGTH	2	Maximum packet length
TCP_FLAGS	1	TCP flags in this flow
ICMP_TYPE	2	The type of the ICMP packet in case this is an ICMP packet

Table 4.1 Netflow 9 fields - Example[6]

4.1.2 KDD99

The KDD99 dataset was generated in 1999 and has proven to be one of the most used training datasets for ML-NIDS. There has been a lot of critique against this dataset, since it is considered to be outdated and inadequate to deal with today's standards and attacks[39][40]. Its features are presented on Table 4.2

Feature	***description***	***type***
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of "wrong" fragments	continuous
urgent	number of urgent packets	continuous

Table 4.2 KDD Features[41]

4.1.3 NSL-KDD

NSL-KDD is an improvement over the KDD99 dataset. Each instance in the training set has information on a single session, which is separated into four sections: a) connection characteristics b) content data c) time-related metrics d) host-based information. NSL-KDD has the following benefits over the KDD dataset[42]:

- Excludes redundant records
- No duplicate records
- Reasonable number of flows

A short list of its features are provided on Table 4.3.

Feature	***description***	***type***
duration	connection duration	continuous
protocol_type	protocol type	discrete
service	targeted network service type	discrete
src_bytes	number of bytes sent from source to destination	continuous
dst_bytes	number of bytes sent from destination to source	continuous
flag	the connection is normal or not	discrete
land	whether the connection is from/to the same host/port	discrete
wrong_fragment	number of “wrong” fragment	continuous
urgent	number of urgent packets	continuous
count	number of connections to the same host in the first two seconds	continuous
serror_rate	“SYN” error on the same host connection	continuous
rerror_rate	“REJ” error on the same host connection	continuous
same_srv_rate	number of of same service connected to the same host	continuous
diff_srv_rate	number of of different services connected to the same host	continuous

Table 4.3 Features of the NSL-KDD Dataset[43]

4.1.4 Kyoto Dataset

There are 24 statistical characteristics in this dataset, 14 of which were extracted using KDD99. There are no packet traces or payload information. The majority of implementations based on this dataset are anomaly-based detection. Some example features are displayed on Table 4.4.

Feature	Description	Type
duration	length (number of seconds) of the connection continuous	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc. discrete	discrete
service	network service on the destination, e.g., http, telnet, etc. discrete	discrete
src_bytes	number of data bytes from source to destination continuou	discrete
IDS_detection	number of triggered alerts	discrete
Malware_detection	type of malware found	string
Label	indicates wether this was an attack	discrete

Table 4.4 Kyoto Dataset Features

4.1.5 UNSW-NB15

The Australian Centre for Cyber Security (ACCS) developed this dataset by collecting tcpdump records of artificial traffic[44][45]. The dataset contains 49 features. ACCS has made all 100GB of data publicly available, including training csv files and the original traffic preserved in pcap files. Its features are presented on Table 4.5.

Feature	Description	Type
srcip	Source IP address	nominal
sport	Source port number	integer
dstip	Destination IP address	nominal
dur	Record total duration	Float
dttl	Destination to source time to live value	Integer
sloss	Source packets retransmitted or dropped	Integer
dloss	Destination packets retransmitted or dropped	Integer
Sload	Source bits per second	Float
Dload	Destination bits per second	Float
Spkts	Source to destination packet count	integer
Dpkts	Destination to source packet count	integer
swin	Source TCP window advertisement value	integer
dwin	Destination TCP window advertisement value	integer

Table 4.5 UNSW-NB15 Dataset Features

4.1.6 MAWILab

The WIDE Project's MAWI Working Group maintains MAWILab's traffic statistics source. The dataset contains a variety of labelled traffic traces that were captured at a variety of

sampling sites. MAWILab[46] is a database that enables academics to examine their methods for detecting traffic anomalies.

4.1.7 CTU MALWARE CAPTURE BOTNET

The CTU-13 dataset is a collection of botnet traffic intercepted at the CTU University in the Czech Republic. The dataset's objective was to collect a huge amount of genuine botnet traffic and make it available for research. The dataset contains both pcap and Packet Flow files.

4.2 ML-NIDS Examples

In this section, we will discuss strategies that leverage established machine learning algorithms such as KNN, Bayesian Networks and Deep Learning in order to create resilient and capable NIDS.

4.2.1 k-Nearest Neighbour IDS

B. Basaveswara Rao and K. Swathi introduced a novel way to attack classification in their publication [47] by combining the kNN, Partial Distance Search kNearest Neighbour (KPDS) [48], and Indexed Partial Distance Search kNearest Neighbor (IKPDS)[49] algorithms. KPDS and IKPDS are both faster versions of kNN. The algorithm is composed of the following steps:

1. The NSL KDD Cup is introduced as input dataset.
2. Data Preprocessing: Input is handled and transformed in a way that is consistent with the kNN approach (normalization, one hot encoding).
3. kNN, KPDS, and IKPDS are used to classify fresh input and predict whether it is an attack ("anomaly" category) or a typical incidence ("normal" class).

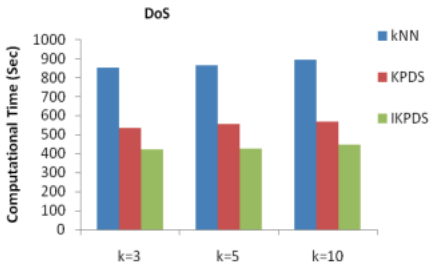


Figure 4.2 Computation time comparison for kNN algorithms [47]

The authors' trials reveal extremely precise outcomes. When the k value is increased, KNN can recognize the majority of incoming threats and improves efficiency (Table 4.6). Additionally, IKPDS and KPDS algorithms have been shown to be significantly faster than the conventional kNN approach (Fig. 4.2).

Table 4.6 kNN-IDS Accuracy for different values of k

k Value	**1**	**2**	**3**
U2R	0.9996	0.9995	0.9994
R2L	0.9988	0.9983	0.9976
DoS	0.9994	0.9991	0.9985

4.2.2 ESIDE-Depian

The ESIDE-Depian software developed by Bringas and Santos[50] is a more sophisticated technique. They use six Bayesian networks to examine packets that enter the system. Three evaluate packets based on their headers (ICMP, TCP, and UDP), while three assess the content and connection (Fig. 4.5).

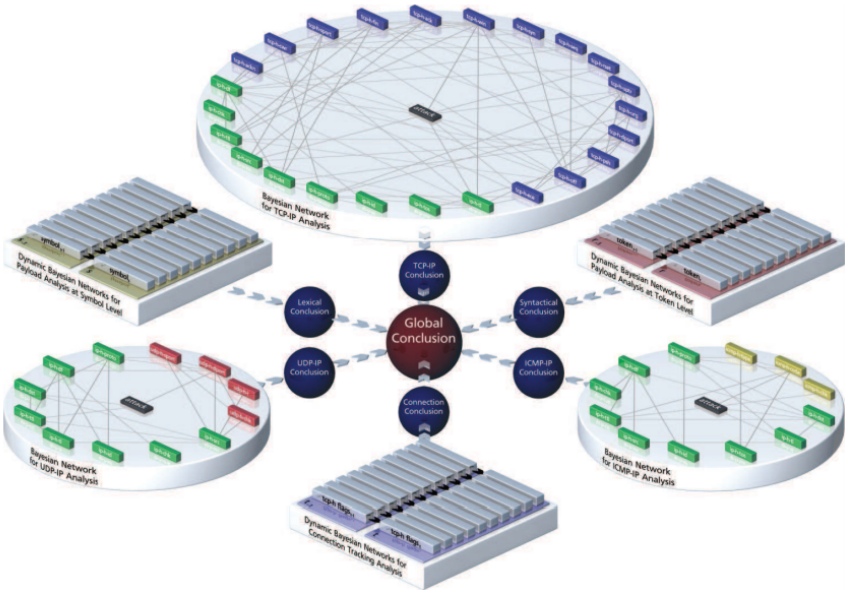


Figure 4.3 ESIDE-Depian Architecture[50]

The training data is derived from datasets created by the ESIDE programmers and consists of both regular and malicious traffic. They classify packets in the dataset as benign or malicious using an IDS called SNORT (Fig. 4.4). Following training, each Bayesian network (expert) attempts to assess the likelihood that arriving packets are part of an attack or intrusion attempt. When a new packet arrives the experts analyze the packet and reach a common verdict regarding the nature of the packet.

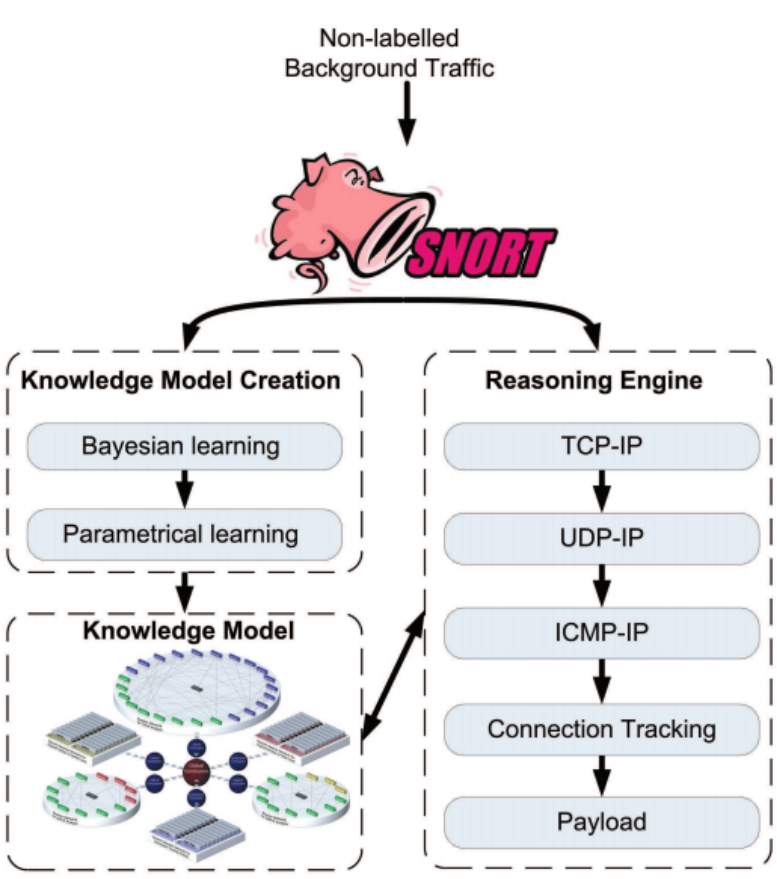


Figure 4.4 Eside's Training Process[50]

Due to the fact that Snort's principal function is to evaluate the headers of incoming packets, the authors compared its efficiency to their ICMP,TCP,UDP-IP expert models. The comparison in Table 4.7 reveals that the findings are equivalent and that both methodologies were successful in tracing all assaults directed at them. Additionally, new fictitious anomalies were created and added into the training traffic in order to assess the ESIDE-Depian Anomaly detection capabilities. Although Snort was unable to identify any of the malicious packets (due to the fact that Snort is a Signature-Based IDS), ESIDE-Depian detected them all. Additionally, despite the fact that Eside-Depian analyzes packets rather than flow data, its Connection Tracking Expert was able to track down all suspicious connection behavior.As a result, ESIDE-Depian is just as effective as Snort at detecting dangerous patterns in packet headers and discovering new threats.

The ESIDE-Depian project served as the initial impetus for the development of the software detailed in this thesis. The Al Pastor software was initially developed with the goal of offering a Deep Learning system capable of analyzing packet headers.

Table 4.7 Eside Results: Connection, Payload Threats Detected

Analysis Type	**Connection Tracking**	**Payload Analysis**
Analysed Packets	226,428	2,676
Attacks in Sample	29	158
ESIDE-Depian Hits	29 (100%)	158 (100%)

4.2.3 Neural Network NIDS

Jia et. al attempted to create a NIDS system by using Neural Networks[51]. They are using the KDD and the NSL-KDD datasets in order to train their models. Their neural network consists of an input layer with 41 neurons (the number of selected features), four hidden layers with RELU activation functions, and an output layer with five neurons that use the SoftMax activation function to classify the packet as an attack type. Finally, they are using the Adam optimization technique for which they found that a Learning Rate of 0.1 produced the best results.

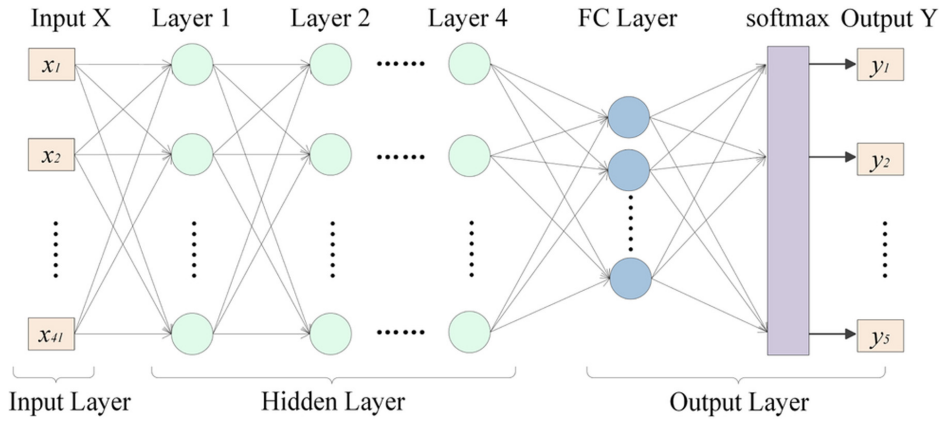


Figure 4.5 Neural Network IDS Solution[51]

To determine the optimal structure for their NN, they conducted multiple tests with varying numbers of neurons and hidden layers. Fig. 4.6 shows the results of their experiments. In general, accuracy increases when using up to three layer and decreases when a fifth layer is added. Their best layout was 4 hidden layers with 100 neurons per layer.

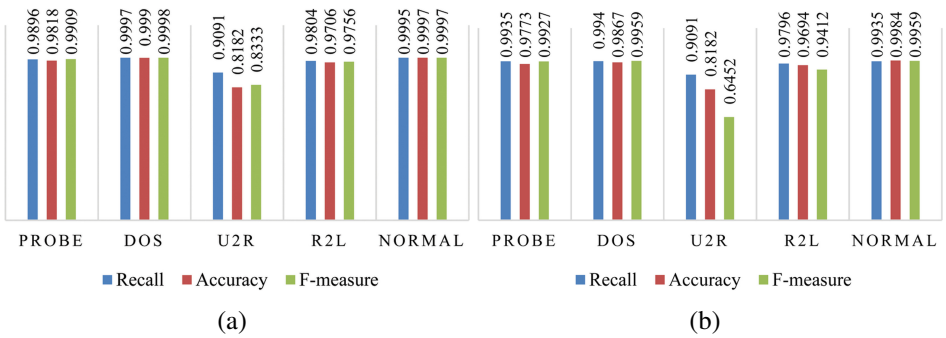


Figure 4.7 NN Performance for different Datasets: (a) KDD (b) NSL-KDD

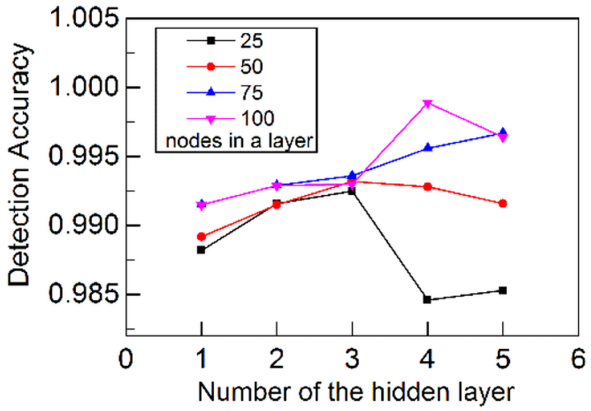


Figure 4.6 Accuracy with regards to Node and Hidden Layer Number[51]

Finally, the results of the NN NIDS are compared to those of other NIDS that employ conventional machine learning approaches, and it is established that the suggested solution IDS outperforms the others in terms of capability and performance. Fig. 4.7 displays the performance of the NN NIDS solution for both the KDD and the NSL-KDD Datasets.

Chapter 5

Al Pastor

The majority of existing solutions of ML-IDS are trained using datasets similar to the ones we presented in the previous chapter. The issue is that when those solutions are used on local or corporate networks, they must be retrained for that specific network (which rarely contains attack samples). Alternatively, a method or a tool must be devised in order to combine training data from existing datasets and the local network in order to generate an average normal state against which to compare anomalous situations.

Additionally, existing datasets contain very little protocol-specific information, and the information included is restricted to protocol headers known to be the source of prior attacks (for example TCP, ICMP flags, number of SYN Packets, etc). On the other hand, while Packet Flow data is particularly effective in tracing a wide variety of attacks, it will have difficulties with attack vectors that do not modify the pace at which data is transmitted but instead rely on changes to the packet headers (for example FREAK, Heartbleed, Downgrade attacks). Such zero-day attacks will be extremely difficult to detect if their fundamental causes are not being monitored in the dataset (for example, a change in a TLS Record or an incorrect bit on the DNS query Type).

This is how Al Pastor came to be. Using packet header data, the ESIDE-Depian NIDS demonstrated how machine learning models (Protocol Experts) may be trained to identify risks. Our first goal was to construct machine learning models capable of detecting attacks in protocols other than those given by ESIDE (TCP, UDP, IP). Our experts were developed using Neural Network approaches and were fairly comparable to those demonstrated on 4.2.3.

However, when developing Al Pastor, it became clear how time-consuming locating the appropriate dataset and testing unique examples may be. Additionally, by observing how several datasets supplied distinct properties, we determined that some form of dataset adaptor would be really beneficial. Given that the majority of publicly accessible datasets

contain their original capture files (pcap), we had the following thought: What if we could construct our own datasets by parsing pcap files? In this manner, we may generate datasets by capturing packets in our local network traffic and also by converting different public datasets into a single format, allowing us to exploit their collective expertise without the need to change our expert models.

As a result, we built a software for producing NIDS datasets from pcap files. Al Pastor supports the creation of two types of datasets:

- *Protocol Header Datasets*: Those dataset contain packet protocol specific information and they differ per protocol stack (for example ETH/IPv4/TCP Dataset, ETH/IPv4/UDP Dataset, ETH/IPv4/QUICC, etc.)

- *Packet-Flow Data*: Netflow-like data similar to the ones described on Section 4.1.1.

A label is then assigned to each entry in the dataset based on whether the respective packets contained threats or raised an Alert. In order to diagnose threats we are using a Signature/Rule-based IDS called Snort[1]. The overall Dataset Creation Process is displayed on Fig. 5.1, a pcap file which contains TCP, UDP and ARP packets is parsed and multiple datasets are created from it. Afterwards Snort assigns a label to each entry of every dataset.

[1] Snort: https://snort.org

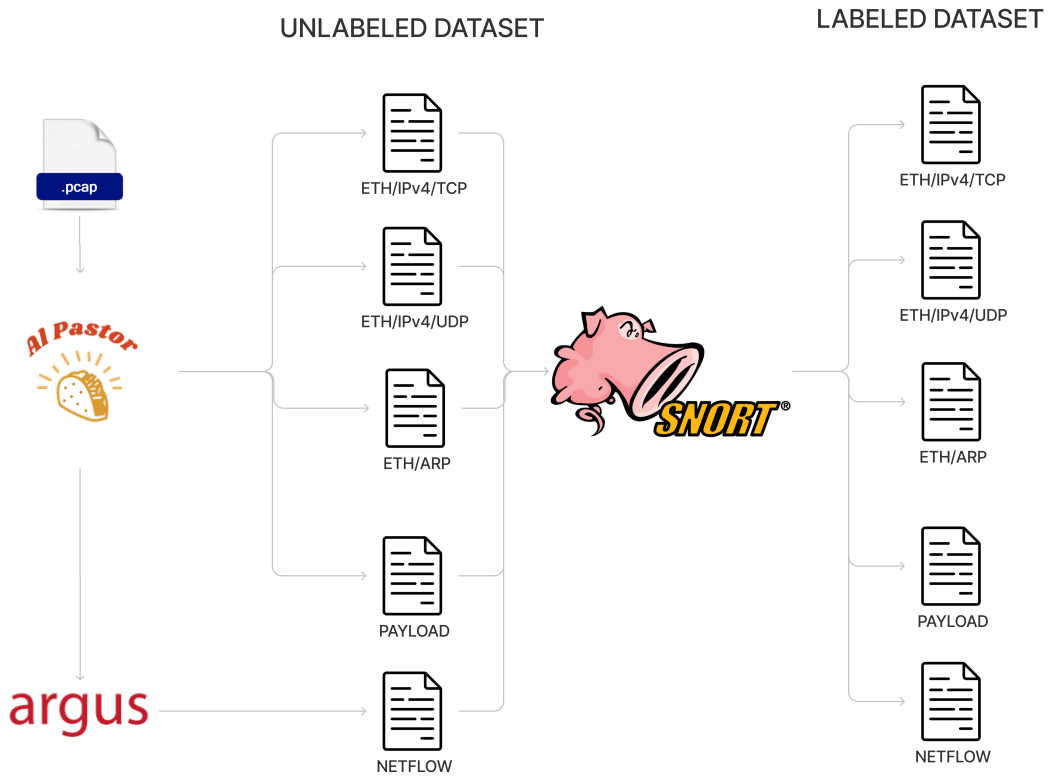


Figure 5.1 Al Pastor: Dataset Creation

The datasets are then utilized to train the NN Experts (Fig. 5.2). Each form of NN Expert enables us to investigate various types of threats:

- *Protocol Header Experts*: These specialists are capable of tracing the same threats that the signature-based IDS can, as well as their variations. Additionally, this IDS functions as an Anomaly-based IDS, defining a typical packet header state and raising warnings when strange packets are identified.
- *Packet-Flow Expert*: Is able to locate weird patterns in rate of data transmissions. Therefore it is mostly able to distinguish Interruption attacks, similarly to IDSs trained with the KDD and the NSL-KDD Datasets

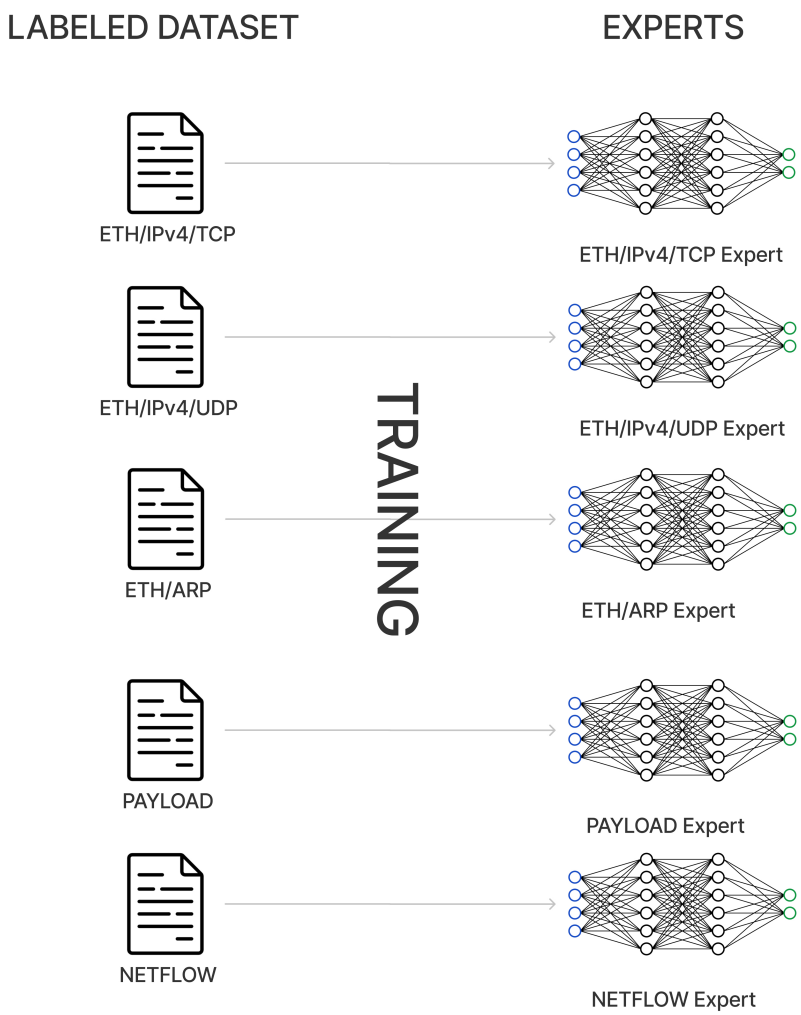


Figure 5.2 Al Pastor: Expert Training

5.1 Toolset

Al Pastor's adoption of only open-source, free-to-use technology was important. As a result of extensive investigation, we decided to use Argus to generate Packet-Flow data, Snort to categorize network traffic, and Tensorflow[2] to construct the Neural Network experts. This section discusses those tools in further detail.

[2]Tensorflow: https://www.tensorflow.org/

5.1.1 Argus

During development, we chose to analyse Packet-Flow datasets as well. As a result, we sought to find software capable of converting pcap files to Flows. We looked at a variety of solutions (such as Zeek or nfdump), but ultimately settled on Argus. Carter Bullard invented Argus, which is practically, the first network flow system, in the early 1980's. The Argus Project is a privately funded open source endeavor focused on proofs of concept. It consists of two components: Argus, a packet processing network flow sensor, and argus-clients, a collection of Argus data processing apps. Fortunately, the Argus Clients Library has a method for converting a pcap file to Netflow-like data. Argus appeared to be the greatest choice due to its illustrious history and straightforward documentation.

5.1.2 Snort

In 1998, Martin Roesch created Snort, a free and open-source network intrusion detection and prevention system that is now maintained by Cisco. It may be used as a packet sniffer or to read data from captured network packets. Snort identifies malicious packets by comparing them against a database of Rules (signatures). While the Enterprise Edition has the most recent Ruleset, the Community Edition is frequently sufficient. Additionally, users have the ability to set their own rules. When Snort is acting as a packet sniffer it has the ability to take actions on packets that it deemed threatening, it can block them, blacklist their entire session, or allow them to pass. In either case, if a packet matched a rule an alert is raised. Snort sets the severity of the packet on a field called priority. They are now arranged in a four-tiered hierarchy. A priority of 1 (extreme) is the most severe, while a priority of 4 (very low) is the least severe.

Snort was chosen as the best signature-based IDS for Al Pastor due to its lengthy history, modular architecture, ease of installation, and large community dataset.

5.1.3 Tensorflow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It offers researchers and developers with a diverse set of tools for building and training their own machine learning models.

Keras is a Tensorflow toolbox for Python that enables the rapid development of Neural Networks via a simple API. Additionally, among Kaggle's top-5 winning teams, Keras is the most often used deep learning framework.

We chose the Tensorflow library because because of its ease of use, extensive documentation, and vibrant community.

5.2 Dataset Generator

The process of creating the dataset is simple:

1. We first create the Packet-Flow Dataset
2. Then we start creating the Protocol Headers Dataset.
3. Every entry of the Protocol Headers Dataset is linked to a Packet-Flow Dataset entry
4. Snort Verdict is assigned to all Datasets

The procedure for developing each dataset is detailed below.

5.2.1 Packet Flow Data

To produce Neflow-like data out of pcap files, the argus tool was utilized. The process of parsing a pcap file with argus requires two steps:

1. Transform the pcap to argus flow data

2. Properly display the argus flow data

In order to transform the pcap to argus flow data, Al Pastor executes the following command:

```
argus -A -J -R -Z -r <pcap file> -w <output argus file>
```

The flags 'A', 'J', 'R', and 'Z' command argus to provide data on byte metrics, packet performance, response times, and packet size. This information is quite valuable for showing the packet flow statistics later on.

Al Pastor utilizes the RA Client Libraries to eventually obtain the packet flow data in the desired manner. The following command parses a file containing argus flow data and outputs it in the format requested:

```
ra -s field_1 -s field_2 ... -s field_n -r packet.argus
```

To guarantee that the features we collected were as efficient as possible, we aimed to collect features that were equivalent to those specified in Mohanad Sarhan et. al. paper "Towards a Standard Feature Set for Network Intrusion Detection System Datasets"[52]. The authors achieved higher DR and F-1 scores while tackling binary and multi-class classification tasks by making minor adjustments to existing datasets and introducing a list of new features.

We utilized the 5-tuple (src/dst ip/port, protocol type) to associate packets with the packet flow to which they belonged. When a packet entry is added to the Protocol Header Dataset, we trace the flow to which it belongs in the Packet-FLow Data and update the packet header dataset's "packetflow id" appropriately. If the packet is determined to be malicious, the record in the Packet-Flow data is changed to reflect the packet's "severity." If a new packet belonging to the same flow is found by Snort to be more severe, the Packet-Flow item is changed to reflect the new value.

The features of the Packet Flow Data are presented on Table 5.1

5.2.2 Protocol Header Datasets

Extraction of the Protocol Header Datasets from the pcap is a straightforward process. We detach each protocol header and extract the data that is relevant to us (the features we have chosen for this protocol). Later, we combine all of the protocol information that we gathered and build a new entry. However, this information will be saved only in the Protocol Dataset associated with this specific protocol stack (For example ETH/IPv4/TCP and ETH/IPv6/TCP are two different protocol stacks and therefore their packets would be stored into separate

Feature	Description	Type
smac	Source MAC addr	nominal
dmac	Destination MAC addr	nominal
rank	Unique record identifier	int
saddr	Source Address (IPv4,IPv6)	nominal
daddr	Destination Address (IPv4,IPv6)	nominal
sport	Source Port (Transport Layer)	int
dport	Destination Port (Transport Layer)	int
proto	Protocol (Top Layer)	nominal
sbytes	#Bytes sent from Source	int
dbytes	#Bytes sent to Source	int
spkts	#Packets sent from Source	int
dpkts	#Packets sent to Source	int
dur	Flow duration	int
state	General transaction state	nominal
flgs	Flow state flags seen in transaction.	nominal
tcpopt	The TCP connection options seen at initiation	nominal
swin	source TCP window advertisement	int
dwin	destination TCP window advertisement	int
synack	Setup round-trip time (SYN, SYNACK)	int
ackdat	Setup round-trip time (SYNACK, ACK)	int
tcprtt	Setup round-trip time (sum of SYNACK, ACK)	int
sload	Source bits per second.	float
dload	Destination bits per second.	float
sttl	src ->dst TTL value	int
dttl	dst ->src TTL value	int
smaxsz	maximum packet size for traffic transmitted by the src	int
sminsz	minimum packet size for traffic transmitted by the src	int
dmaxsz	maximum packet size for traffic transmitted by the dst	int
dminsz	minimum packet size for traffic transmitted by the dst	int
sappbytes	src ->dst application bytes	int
dappbytes	dst ->src application bytes	int
sretrans	source pkts retransmitted	int
dretrans	destination pkts retransmitted	int
pretrans	percent pkts retransmitted	int
psretrans	percent source pkts retransmitted	int

Table 5.1 Al Pastor: Packet-Flow Data Features

Datasets). A simplistic form of the Dataset extraction algorithm is presented bellow in Python.

```
datasets = {}
for packet in pcap_packets:
    packet_protos = ""
    packet_features = []
    for header in get_headers(packet):
        proto = get_proto_type(header)
        # get dataset features
        features = get_features(header)
        # join with features list
        packet_features = [*packet_features, *features]
    # if there has never been a similar dataset created before
    # generate a new one
    if packet_protos not in datasets:
        datasets[packet_protos] = []
    datasets[packet_protos].append(packet_features)
```

For TLS (and IPv6) packets, where the packet headers may be composed of many sub headers (records for TLS, next headers for IPv6), we regard each underlying mini-header as a separate protocol. For instance, a TLS packet containing three Records: TLS_SERVER_HELLO, TLS_CHANGE_CIPHER, TLS_DATA will be added to the Dataset:

$$ETH/IPvX/TCP/TLS\_SERVER\_HELLO/TLS\_CHANGE\_CIPHER/TLS\_DATA$$

The features that have been selected for each protocol are listed in Table 5.2.

5.3 Neural Network Models

Al Pastor creates a unique Neural Network for each dataset, and the designs of these networks may vary, we will see how each of this performs in the last section. However there are some key architectural decisions which will be described bellow.

5.3.1 Data Pre-Processing

Al Pastor employs One Hot Encoding to convert nominal data to numerical data. Because each feature in the Protocol Header dataset is instantly converted to a number during processing of the pcap file, we need to apply the OHE only to some featurs of the Packet-Flow Dataset ("proto", "state", "flgs", "tcpopt"). Then we normalize the rest of the data as described in section 3.1.3.

proto	**features**
eth	src, dst, type
arp	hw_type, proto_type, hw_size, proto_size, opcode, is_unicast
dhcp	hw_type, hw_len, hops, secs, flags_bc, flags_reserved, option_type, option_length, option_value, option_dhcp, option_request_list_item, option_end
dns	flags_response, flags_opcode, flags_truncated, flags_recdesired, flags_z, flags_checkdisable, count_queries, count_answers, count_auth_rr, count_add_rr, qry_class, qry_name_len, count_labels, qry_type, resp_class, resp_len, resp_ttl, resp_type, response_to
ipv4	src, dst, checksum_status, ttl, proto, flags_rb, flags_df, flags_mf, frag_offset, dsfield_dscp, dsfield_ecn, hdr_len, len
ipv6	tclass, tclass_dscp, tclass_ecn, plen, next, hlim, flow
icmp	code, type, checksum_status
udp	srcport, srcport, len, time_relative, time_delta
tcp	srcport, dstport, len, seq, ack, nxtseq, hdr_len, flags_res, flags_ns, flags_cwr, flags_ecn, flags_urg, flags_ack, flags_push, flags_reset, flags_syn, flags_fin, window_size_value, window_size_scalefactor, checksum_status, urgent_pointer, time_relative, time_delta,option_kind, option_len, options_mss_val, options_wscale_shift, options_wscale_multiplier, options_timestamp_tsval, options_timestamp_tsecr, options_sack_perm, options_eol, options_nop
tls	record_type, record_length, record_version, handshake_type, handshake_length, handshake_version, handshake_random, handshake_session_id_length, handshake_cipher_suites_length, handshake_ciphersuite, handshake_comp_methods_length, handshake_comp_method, handshake_extension_type, handshake_extension_len, handshake_extensions_supported_groups_length, handshake_extensions_supported_group, handshake_extensions_ec_point_formats_length, handshake_extensions_ec_point_format, handshake_sig_hash_alg_len, handshake_sig_hash_alg, handshake_extensions_key_share_client_length, handshake_extensions_key_share_group, handshake_extensions_key_share_key_exchange_length, handshake_extensions_supported_versions_len, handshake_extensions_supported_version, handshake_server_curve_type, handshake_server_named_curve, handshake_server_point_len, handshake_session_ticket_length, handshake_session_ticket_lifetime_hint

Table 5.2 Al Pastor: Protocol Headers Dataset

We subsequently split the initial dataset into two subsets: one for testing and one for training (0.2 and 0.8 of the original size respectively).

5.3.2 Adding Noise

However, there is an issue that we must address. Each neural network is trained using the input data to distinguish between the Snort-generated attack types (1–4). This means that our algorithm can distinguish between packets that "appear to be" typical network packets (category 0) and packets that "appear to be" Snort-detected assaults. However, what happens if a packet that is part of a new assault does not fit into any of the categories?

The package will then be classified according to whatever category it most closely resembles. However, it is quite feasible that this packet was irrelevant in prior assaults and hence classified as a "normal" packet. As a result, it is evident that we want a new category, one that will assist us in identifying anomalous packets, or more precisely, abnormal network activity. To do this, we introduce noise into the training data for each dataset (i.e., packets with random values in fields) and assign them a class 5. From now on, packets that Al Pastor identifies as category 5 are packets that do not conform to the network's regular behavior.

5.3.3 NN Architecture

The produced NN is optimized using the Adam method; in the beginning, a callback function attempts to discover the ideal learning rate for each dataset.

Each neuron in the input and hidden layers is stimulated using the RELU activation function, which was chosen for its simplicity, reliability, and rapid convergence during training. Due to the fact that we want to be able to categorize packets based on their "Severity" level, the output layer has just four neurons that are triggered using softmax, which is the de facto activation function for classification issues. During the Network's training phase, we introduced an accuracy monitor with a 20-round patience. If the accuracy does not improve for at least 20 consecutive rounds, we return the weights to the optimal Neural Network Configuration. Otherwise, 100 rounds of training are conducted. Appendix B contains more information about the architecture and implementation of the NN models.

5.4 Results

Pcap Files from the MawiLab4.1.6 and CTU-Botnet4.1.7 datasets were used in the following experiments. In addition, simulated network traffic was utilized to demonstrate different

elements of the Al Pastor. The training datasets are around 80% of the original size, whereas the testing datasets are 20% of the original size.

5.4.1 NN Configuration

We had specified the technology for our Neural Networks from the outset. RELU activation functions are employed in the Hidden Layers neurons, as well as SoftMax on the output. To train the model, the Adam Optimization Algorithm would be employed. However, because we had not yet settled on the topology of the Neural Network, we needed to conduct some experiments to determine the ideal number of neurons and layers.

We attempted to increase the number of nodes and hidden layers in our NN repeatedly using data from the MAWI dataset in order to determine the configuration with the best potential accuracy. We first doubted our efforts would succeed due to the fact that we were essentially comparing separate datasets to different Neural Network Models, but in the end, we observed some trends in the findings. It seems as though the ideal implementations always featured 60 neurons and that between four and six hidden layers generated the greatest outcomes. This is evident not only when testing different configurations for the Protocol Header Datasets (Fig. 5.3) but also when inspecting the Packet Flow Dataset (Fig. 5.4).

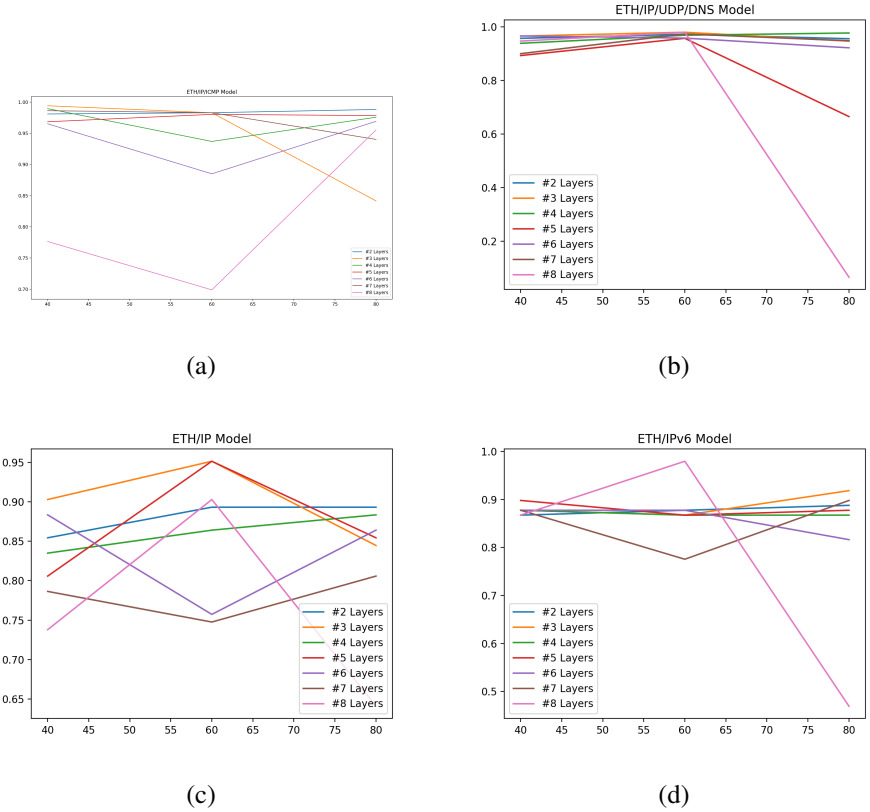


Figure 5.3 Protocol Header Experts Configuration: Selecting the best #Nodes/#Layers Combination

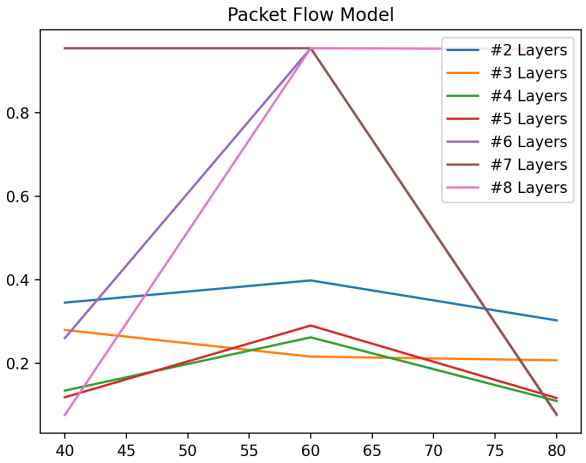


Figure 5.4 Flow Expert Configuration: Selecting the best #Nodes/#Layers Combination

Additionally, we needed to choose a suitable number for the Adam algorithm's learning rate. Once again, we utilized numerous rounds to determine the optimal learning rate for our data. Similarly to our prior testing, we noticed that the majority of datasets had a similar pattern of behavior, with the best learning rate being about 0.016. Figures 5.5 and 5.6 display the test results for the Protocol Header and Packet Flow Experts respectively.

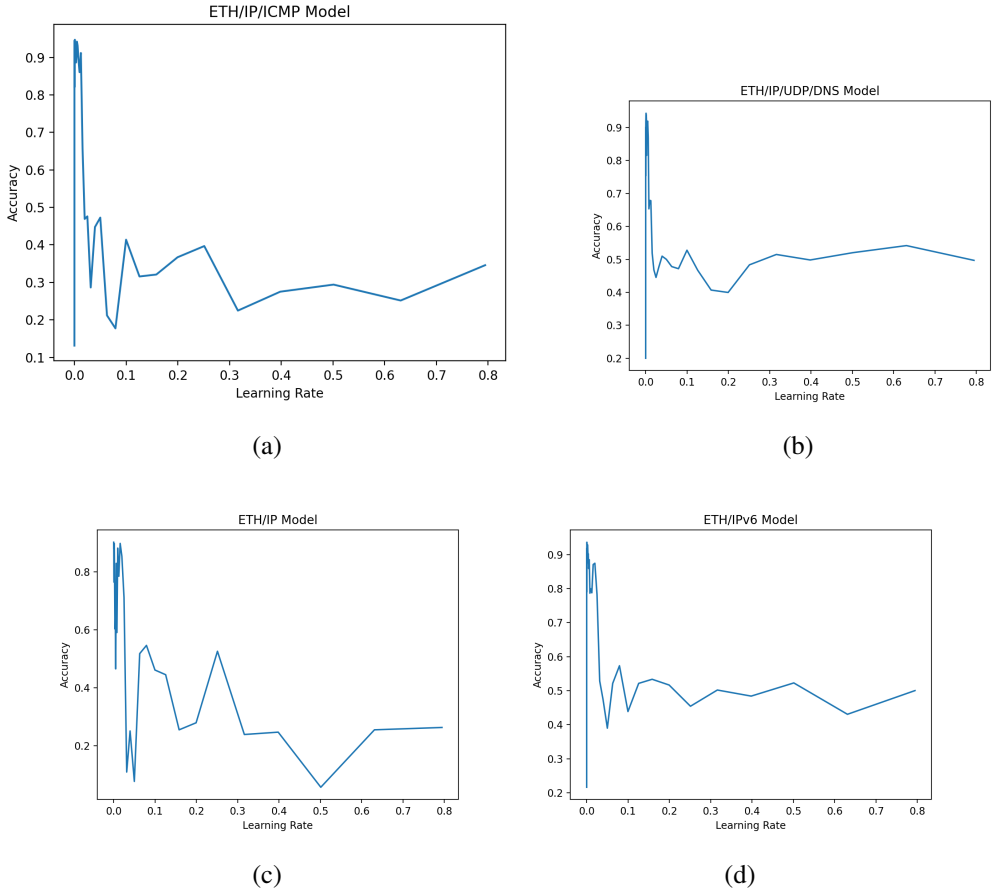


Figure 5.5 Protocol Header Experts Configuration: Selecting the best Learning Rate

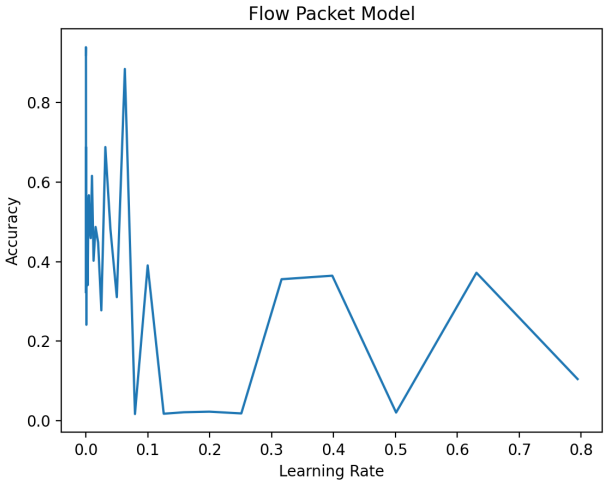


Figure 5.6 Packet Flow Expert Configuration: Selecting the best Learning Rate

According to the preceding, all Neural Network experts have six hidden layers and the Adam's Optimization Algorithm's Learning Rate is set at 0.0016.

5.4.2 Testcase: DNS Response Headers

To begin, we will demonstrate how adding more protocol data affects the NN model's efficiency. A Command and Control attack (CNC), in general, creates a control link between an attacker and a target. Snort can usually detect these sorts of attacks by examining the outgoing and incoming packet rates. However some CNC attacks can be discovered by investigating DNS packets that contain more replies than questions, as well as DNS packets with extremely short time-to-live values.

As a consequence, we generated network traffic composed entirely of DNS requests, half of which displayed features of CNC attacks. Three datasets were produced from the network traffic: 1) the first included a subset of fields of the DNS queries 2) the second included fields from the DNS Responses (such as minimum TTL), and 3) the third one just included Packet Flow data.

It is theoretically impossible to relate the supplied Packets in the dataset to any threats by training the NN on the first dataset (except if it manages to find out some hidden connection that we were unaware of), since the dataset does not contain any DNS Response data. Fig. 5.7 illustrates the dataset's results. The dataset performed poorly in identifying threats of type 1 (CNC) as it achieved 35% Detection Rate.

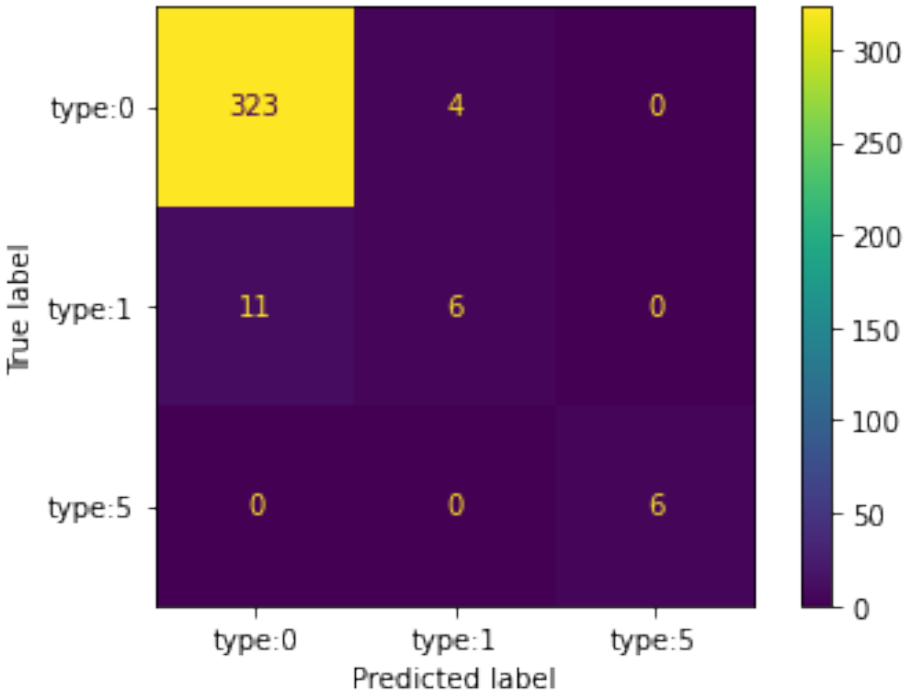


Figure 5.7 CNC-Attack: Missing DNS Response Dataset

However, when we train our NN models on the second dataset, we obtain a 100% Detection Rate (Fig. 5.9). It is very feasible that our model discovered a way to correlate DNS Response data (TTL and number of DNS Responses) to CNC attacks. However, the FAR increases to 4%. This might be because certain packets with a short TTL or many DNS responses are wrongly identified as malicious. This is a very tiny percentage of packets, however, and should not be reason for alarm.

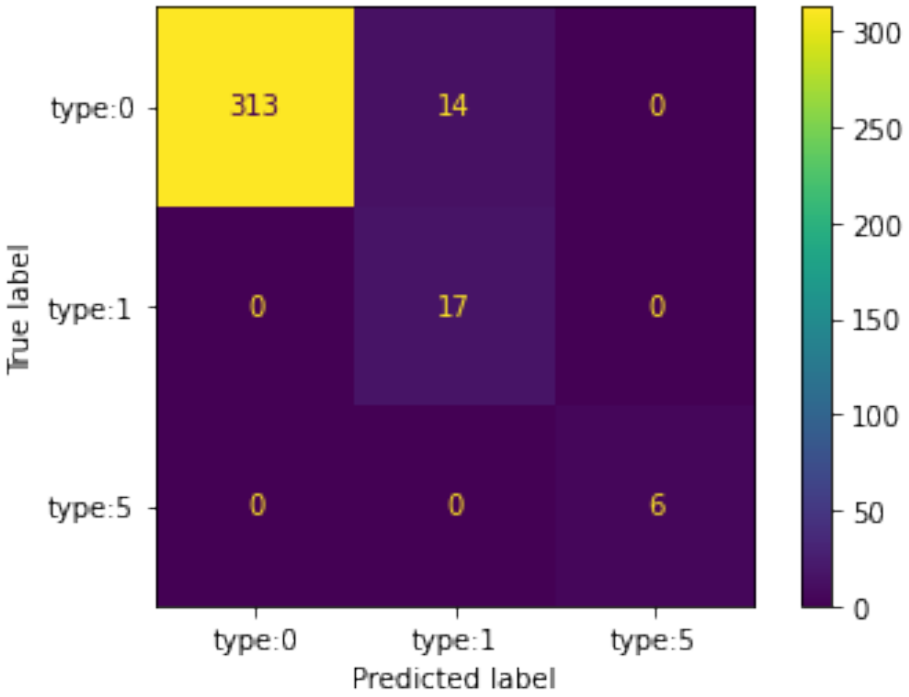


Figure 5.8 CNC-Attack: Full DNS Header Dataset

In the last test, we used the Packet Flow dataset to train the Neural Network (Fig. 5.9). Unfortunately, the dataset lacks the essential information to estimate the CNC threat since it does not contain any information on DNS packets; instead, it examines rate data (packet flows). Given its low Detection Rate and the fact that it incorrectly labeled certain Normal packets as CNC attacks, we may surmise that the Neural Network may have associated some irrelevant features in the dataset with type 1 attacks.

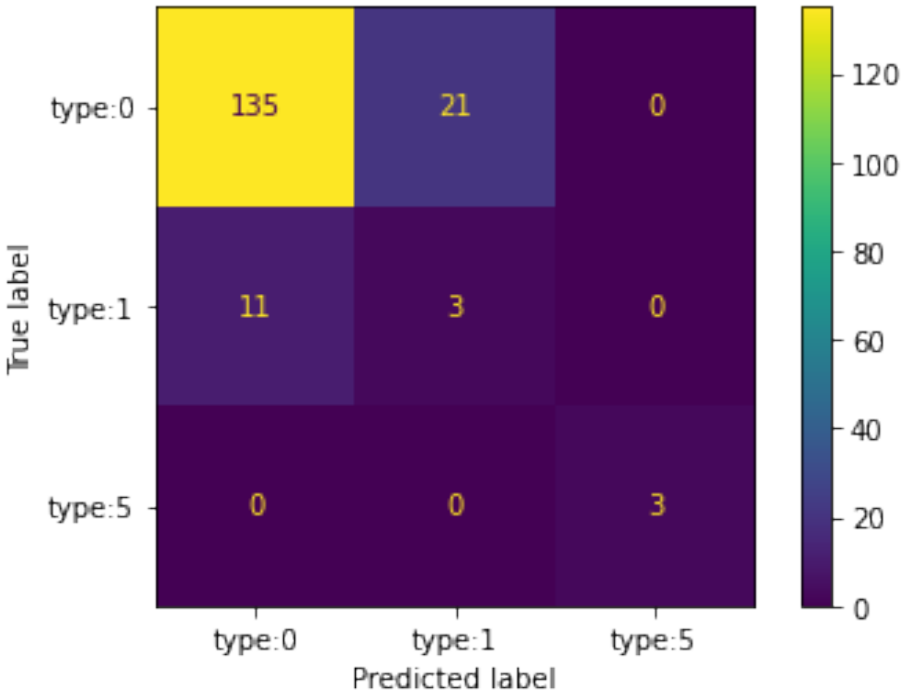


Figure 5.9 CNC-Attack: Packet Flow Model

To identify diverse risks, we conclude that we must extract as much information as possible from packet headers. This, however, adds to the processing time. Generally, a trade-off occurs between accuracy and calculation time.

5.4.3 Testcase: BotNet Attack

In the following test, we attempted to demonstrate that the two types of datasets (Payload and Packet flow) are capable of diagnosing distinct sorts of intrusions and that we should use both datasets to more accurately identify threats.

To substantiate our assertions, we used portions of the CTU Malware Dataset (from the Neris Botnet scenario). Our sample contains 85097 DNS packets, 34428 of which were associated with a botnet attack. Snort recognized the majority of assaults based on the rate of incoming packets. These were often Command and Control attacks, which Snort classified as category 1 attacks (the most dangerous).

We therefored proceed to train two NN models, the ETH_IP_DNS Model and the packet flow model. Fig. 5.10 displays the results of the Packet Header Model. Snort identified port scan efforts as category 3 (by identifying the rate of incoming messages), whereas crush attempts on Recursive DNS resolvers were classed as category 2. As previously noted, CNC's excessive DNS requests were tagged as 1. The ETH_IP_DNS Model is capable of

identifying attacks of type 2 and 3, but it is unable to identify attacks of type 1 due to the model's lack of packet data rate monitoring. The large FAR implies that the model was unable to fully correlate any of the attacks with the header data.

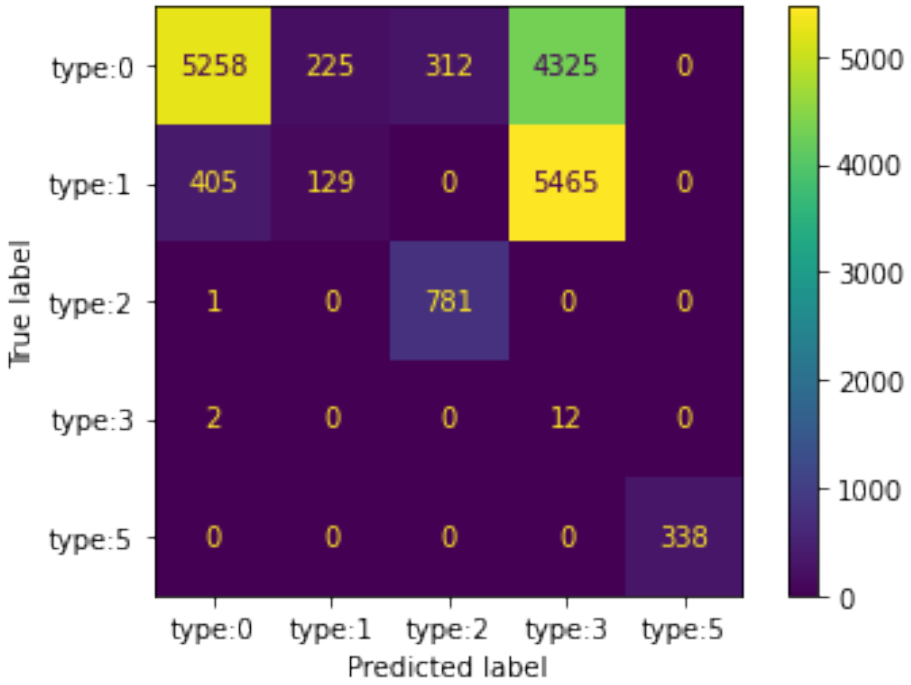


Figure 5.10 Neris Botnet: ETH_IP_DNS Model

When we instead use the Packet Flow Dataset (Fig. 5.11), we observe superior results. To begin, it is self-evident that this dataset contains no assaults of type 3. This is because the majority of those assaults were carried out in conjunction with type 1 attacks inside the same flow. Because category 1 attacks are deemed to be more serious, the corresponding packet flows are classified as category 1 rather than category 3. In this test, the model maintained a low FAR (2%), while attempting to trace almost all packet flows attempting to conduct a DoS Attack (95% percent DR). This was to be expected, as packet flows retain information about data exchange rates, which enables us to identify DoS assaults. Regrettably, the model proved incapable of tracing type 2 assaults (since it did not have access to header data information). In this case, we would be able to trace all incoming threats by utilizing both the Packet Flow expert's and the appropriate Protocol Header expert's verdicts.

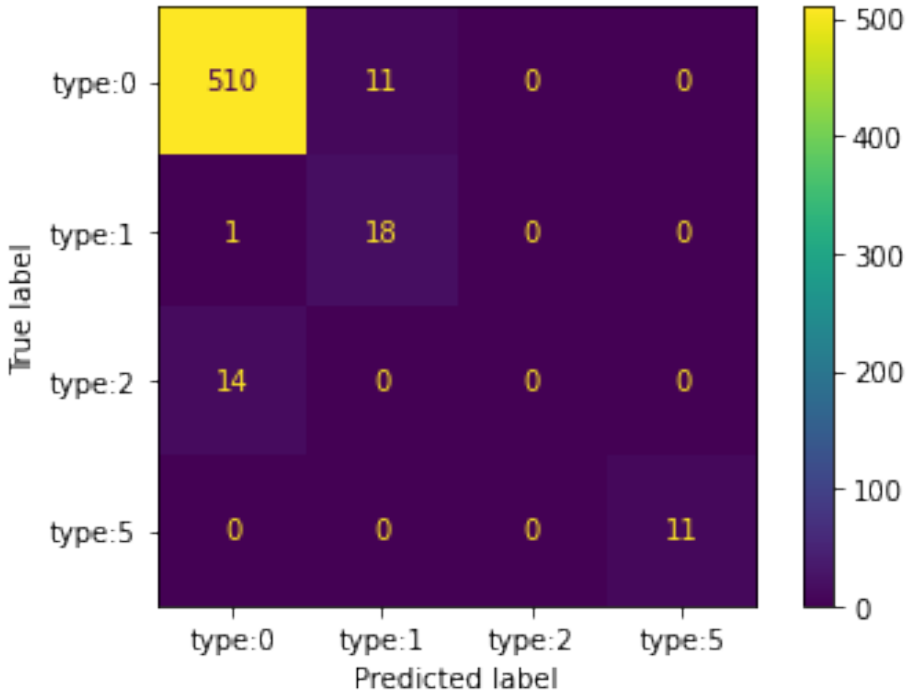
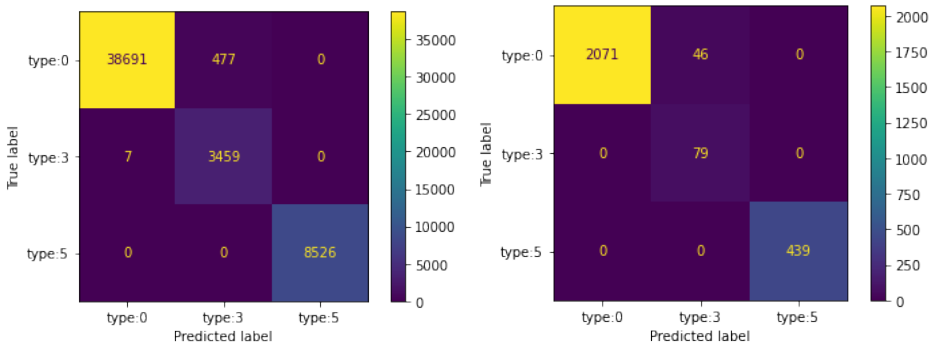


Figure 5.11 Neris Botnet: Packet Flow

5.4.4 Al Pastor as an Signature-Based IDS

To demonstrate Al Pastor's efficacy against previously identified threats, we ran it on TCP and ICMP datasets acquired from MAWI. Their performance is displayed in Fig. 5.12.



(a) (b)

Figure 5.12 Al Pastor: Traffic analysis against different Protocol types a) TCP b) ICMP

Al Pastor's efficiency is obvious, and its performance is comparable to Snort's. Thus, Al Pastor has the ability to accomplish everything Snort does (Table 5.3). However, Al Pastor

should ideally be capable of more: it should be capable of diagnosing previously unknown attacks.

	TCP	ICMP
Total Packets	51160	2635
Attacks In Sample	11985	518
Threats Discovered	11978	518
Detection Rate	99.9%	100%
False Alarm Rate	1.2%	2.2%
F1-Score	97.9	95.8

Table 5.3 Al Pastor: MAWI Dataset ICMP-TCP Traffic Analysis

5.4.5 New Threats/Anomalies

Thus far, we have established Al Pastor's effectiveness against recognized threats. However, that is precisely what a signature-based IDS does. We did further experiments to demonstrate that Al Pastor can act as an anomaly-based detection IDS as well. During those experiments we captured some typical traffic and then introduced some abnormal packets that the system had never seen before. In order to generate packets Python's Scapy library was used [3].

Malformed ARP

We began the test by collecting data from the Mawi Dataset and training our Protocol Header Experts on it. After the training was complete, we produced eight ARP packets with the field OP set to a number between 100 and 1000 (the operation field in ARP is always set to 1 or 2) and sent it to the network.

```
>>> sendp(Ether(dst="ff:ff:ff:ff:ff:ff",src="00:11:22:33:44:55")/ARP(
    hwsrc="00:11:22:33:44:55",op=random.randint(100,1000),pdst="
    132.11.44.2"))
```

We saved the collected traffic in pcap format and sent it to Al Pastor for conversion to the Protocol Header Experts' preferred data format. During this conversion, Al Pastor also assigned Snorts' verdict on every packet. In the end we used our ETH_ARP expert to predict the Category of each of the given ARP packets. We compared the prediction with Snorts' output and the results are presented on Figure(5.13).

[3] Scapy: https://scapy.net/

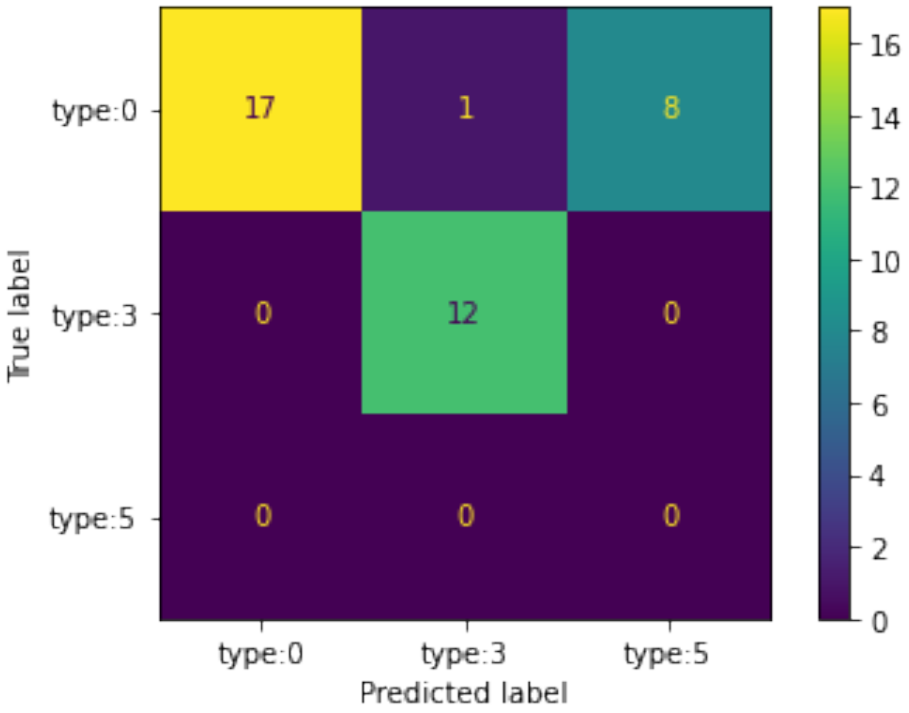


Figure 5.13 Outsnorting Snort: Malformed ARP Packet

The model properly identified all category 3 packets (included in the traffic we monitored). However, it made the false assumption that one innocuous packet was of category 3. Finally, the Packet Header Expert was able to track eight suspicious packets (category 5) that elicited no Snort alarms. Naturally, those were the faulty packets we produced.

Malformed TCP

The ETH_IP_TCP Protocol Header Expert in this experiment has been trained by using an excerpt of 213000 TCP packets from the MAWI Dataset. To demonstrate our approach, we wanted to generate a packet that would be distinct from the TCP payloads that the expert is accustomed to and yet would not trigger a Snort alert. The packet we created was a TCP SYN Request with the TCP Reserved fields set to a random value.

```
>>> sendp(Ether(src="ab:ab:ab:ab:ab:ab", dst="cd:cd:cd:cd:cd:cd")/IP(
    src="192.187.122.1", dst="192.111.1.3")/TCP(sport=12345, dport
    =54321,flags="S",reserved=123))
```

We sent six of these messages to randomly chosen destination addresses and then repeated the steps outlined at 5.4.5. The pcap file created contained 16 innocuous packets in addition to our six faulty ones. When presented with the exported dataset, the ETH_IP_TCP Expert accurately classified all packets (Fig. 5.14).

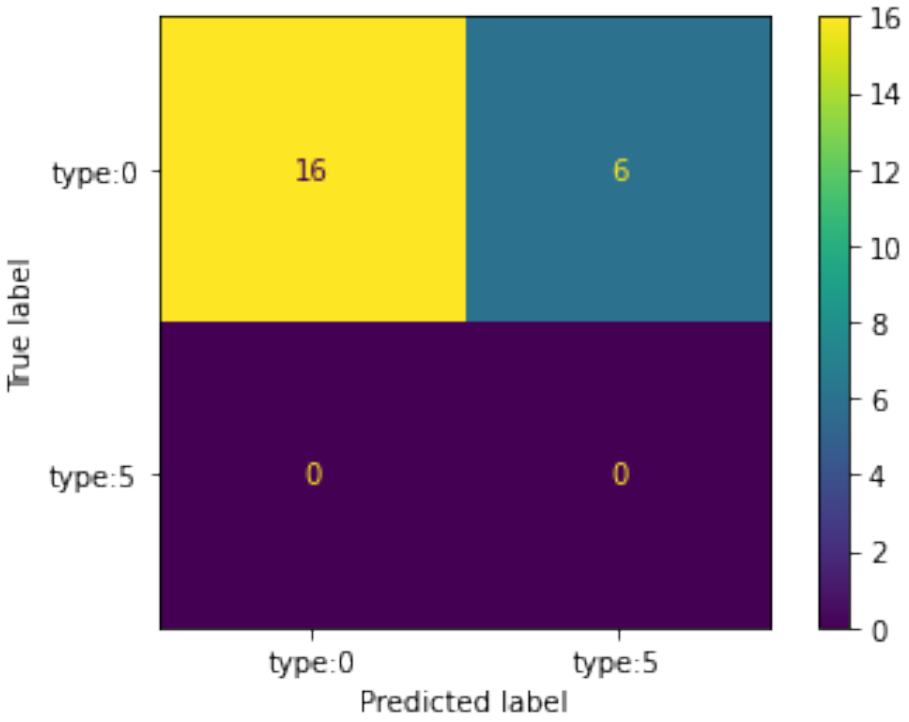


Figure 5.14 Outsnorting Snort: Malformed TCP Packet

This experiment reveals that Al Pastor is capable of tracing strange, potentially harmful packets (while typical signature-based IDSs such as Snort ignore them), hinting that Al Pastor can act as an anomaly-based intrusion detection system.

Conclusion

Although security is unpredictable, since a danger cannot be predicted (if it could, it would not be a threat), enforcing effective and thorough security monitoring makes the system less susceptible to attacks. Even if a new danger is introduced (a new zero-day exploit), we must be able to reassure future technology users that the system is capable of adapting and overcoming the problem. Because it is our obligation to reassure consumers that, even if a hypothetical security breach has catastrophic effects (for example, automobile and medical equipment hacking), they should feel secure while taking use of contemporary technology's many advantages and conveniences.

This thesis provides a thorough examination of network intrusion detection systems based on ML and DL algorithms. ML and DL approaches are addressed and briefly detailed, as are IDS implementations that aid in the identification of threats in a network system. Finally, a novel method of evaluating network data is shown by merging protocol-related and netflow datasets. The aforementioned system, Al Pastor, was designed and developed during the duration of this thesis and its results are also presented here.

While Al Pastor may not comprehend how each assault would cause harm, it is one step closer to identifying new threats by evaluating not just the system's flow data but also doing packet inspection. Al Pastor is capable of recognizing unexpected traffic patterns even when they have never been observed before.

There are several suggestions for future enhancements to Al Pastor. Among them are the following:

- Supplementing the protocol header datasets with statistics (often found in Netflow files) and finally attempting to merge the two types of datasets into one.
- Completing the project and developing a stand-alone IDS system that can be deployed in real time alongside a Firewall in corporate and household networks to train and assess incoming packets.
- Increase the number of parsers for additional protocols.

- Developing effective methods for analyzing encrypted packets
- More testing against larger samples of genuine threats and abnormal packets, which should result in alarms, system performance review, and maybe additional fixes.

List of Figures

List of Tables

Glossary

Roman Symbols

AI Artificial Intelligence

AIDS Anomaly-based Intrusion Detection System

DR Detection Rate

FAR False Alarm Rate

FN False Negative

FP False Positive

HIDS Host Intrusion Detection System

IDS Intrusion Detection System

IOC Indicators of Compromise

IoT Internet Of Things

IoV Internet Of Vehicles

IPS Intrusion Prevention System

ML-NIDS Machine Learning - Network Intrusion Detection Systems

ML Machine Learning

NIDS Network Intrusion Detection System

NN Neural Network

OHE One Hot Encoding

Superscripts

RNN Recurrent Neural Networks

TNR True Negative Rate

TN True Negative

TP True Positive

Bibliography

[1] A. L. Samuel, Some studies in machine learning using the game of checkers, IBM Journal of Research and Development 3 (3) (1959) 210–229. doi:10.1147/rd.33.0210.

[2] M. S. Campbell, A. J. Hoane, Search control methods in deep blue, in: In AAAI Spring Symposium on Search Techniques for Problem Solving Under Uncertainty and Incomplete Information, AAAI Press, p. pages.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489. doi:10.1038/nature16961.
URL https://doi.org/10.1038/nature16961

[4] B. Marr, The five biggest cyber security trends in 2022, https://www.forbes.com/sites/bernardmarr/2021/ 12/17/the-five-biggest-cyber-security-trends-in-2022/?sh=207f1f744fa3.

[5] H. Liu, B. Lang, Machine learning and deep learning methods for intrusion detection systems: A survey, Applied Sciences 9 (20) (2019) 4396. doi:10.3390/app9204396.
URL http://dx.doi.org/10.3390/app9204396

[6] Cisco it case study: Netflow, https://www.cisco.com/c/dam/en_us/about/ciscoitatwork/downloads/ciscoitatwork/pdf/Cisco_IT_Case_Study_Netflow_print.pdf.

[7] L. Mayer Lux, Defining cyberterrorism, Revista chilena de derecho y tecnologÃa 7 (2018) 5 – 25.
URL http://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0719-25842018000200005&nrm=iso

[8] I. Winkler, A. T. Gomes, Chapter 2 - cyberwarfare concepts, in: I. Winkler, A. T. Gomes (Eds.), Advanced Persistent Security, Syngress, 2017, pp. 15–19. doi:https://doi.org/10.1016/B978-0-12-809316-0.00002-6.
URL https://www.sciencedirect.com/science/article/pii/B9780128093160000026

[9] H. Chwe, The rise of cyber warfare: The digital age and american decline, 2016.

[10] D. Bohn, Us cyberattack reportedly hit iranian targets, https://www.theverge.com/2019/6/22/18714010/us-cyberattack-ira nian-targets-missile-command-report (Jun 2019).

[11] P. Singer, Stuxnet and its hidden lessons on the ethics of cyberweapons, Case Western Reserve Journal of International Law 47 (2015) 79.

[12] W. Stallings, Cryptography and Network Security: Principles and Practice, 1998.

[13] A. Tyagi, Tcp/ip protocol suite, International Journal of Scientific Research in Computer Science, Engineering and Information Technology (2020) 59–71doi:10.32628/CSEIT206420.

[14] B. E. Carpenter, Architectural Principles of the Internet, RFC 1958 (Jun. 1996). doi:10.17487/RFC1958.
URL https://www.rfc-editor.org/info/rfc1958

[15] G. Genosko, The case of 'mafiaboy' and the rhetorical limits of hacktivism, Fibreculture Journal (01 2006).

[16] F. Yihunie, E. Abdelfattah, A. Odeh, Analysis of ping of death dos and ddos attacks, 2018, pp. 1–4. doi:10.1109/LISAT.2018.8378010.

[17] M. Bogdanoski, T. Shuminoski, A. Risteski, Analysis of the syn flood dos attack, International Journal of Computer Network and Information Security 5 (2013) 1–11. doi:10.5815/ijcnis.2013.08.01.

[18] S. Suroto, A review of defense against slow http attack, JOIV : International Journal on Informatics Visualization 1 (2017) 127. doi:10.30630/joiv.1.4.51.

[19] M. Williams, M. Tüxen, R. Seggelmann, Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension, RFC 6520 (Feb. 2012). doi:10.17487/RFC6520.
URL https://www.rfc-editor.org/info/rfc6520

[20] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, J. A. Halderman, The matter of heartbleed, in: Proceedings of the 2014 Conference on Internet Measurement Conference, IMC '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 475–488. doi:10.1145/2663716.2663755.
URL https://doi.org/10.1145/2663716.2663755

[21] S. Lee, Y. Shin, J. Hur, Return of Version Downgrade Attack in the Era of TLS 1.3, Association for Computing Machinery, New York, NY, USA, 2020, p. 157–168.
URL https://doi.org/10.1145/3386367.3431310

[22] M. Green, Attack of the week: Freak (or 'factoring the nsa for fun and profit'), https://blog.cryptographyengineering.com/2015/03/03/attack-of-week-freak-or-factoring-nsa/.

[23] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, E. Vázquez, Anomaly-based network intrusion detection: Techniques, systems and challenges, Comput. Secur. 28 (1–2) (2009) 18–28. doi:10.1016/j.cose.2008.08.003.
URL https://doi.org/10.1016/j.cose.2008.08.003

[24] J. Neyole, Y. Muchelule, A review of intrusion detection systems, International Journal of Computer Science and Information Technology Research 5 (2017) 1–5.

[25] D. Stiawan, H. Abdullah, Y. Idris, Characterizing network intrusion prevention system, International Journal of Computer Applications 14 (01 2011). doi:10.5120/1811-2439.

[26] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, M. Fischer, Taxonomy and survey of collaborative intrusion detection, ACM Computing Surveys 47 (05 2015). doi:10.1145/2716260.

[27] Chapter 1 - introduction to intrusion detection systems, in: J. Burton, I. Dubrawsky, V. Osipov, C. Tate Baumrucker, M. Sweeney (Eds.), Cisco Security Professional's Guide to Secure Intrusion Detection Systems, Syngress, Burlington, 2003, pp. 1–38. doi:https://doi.org/10.1016/B978-193226669-6/50021-5.
URL https://www.sciencedirect.com/science/article/pii/B9781932266696500215

[28] T. M. Chen, P. J. Walsh, Chapter 3 - guarding against network intrusions, in: J. R. Vacca (Ed.), Network and System Security (Second Edition), second edition Edition, Syngress, Boston, 2014, pp. 57–82. doi:https://doi.org/10.1016/B978-0-12-416689-9.00003-4.
URL https://www.sciencedirect.com/science/article/pii/B9780124166899000034

[29] M. Millett, Steven; Toolin, J. Bates, Analysis of computer audit data to create indicators of compromise for intrusion detection, SMU Data Science Review 2 (1) (2019).

[30] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, E. Vázquez, Anomaly-based network intrusion detection: Techniques, systems and challenges, Computers Security 28 (2009) 18–28. doi:10.1016/j.cose.2008.08.003.

[31] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, Survey of intrusion detection systems: techniques, datasets and challenges, Cybersecurity 2 (12 2019). doi:10.1186/s42400-019-0038-7.

[32] D. G. Kumar Ahuja, Evaluation metrics for intrusion detection systems-a study, International Journal of Computer Science and Mobile Applications 11 (06 2015).

[33] J. Ulvila, J. Gaffney, Evaluation of intrusion detection systems, Journal of Research of the National Institute of Standards and Technology 108 (2003) 453. doi:10.6028/jres.108.040.

[34] J. Joyce, Bayes' Theorem, in: E. N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, Fall 2021 Edition, Metaphysics Research Lab, Stanford University, 2021.

[35] M. Nielsen, Neural networks and deep learning, http://neuralnetworksanddeeplearning.com/chap1.html.

[36] D. Kingma, J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations (12 2014).

[37] J. Guo, Backpropagation through time, 2013.

[38] Cisco ios netflow version 9 flow-record format - white paper., https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.pdf.

[39] A. Divekar, M. Parekh, V. Savla, R. Mishra, M. Shirole, Benchmarking datasets for anomaly-based network intrusion detection: Kdd cup 99 alternatives, in: 2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS), 2018, pp. 1–8. doi:10.1109/CCCS.2018.8586840.

[40] P. Aggarwal, S. K. Sharma, Analysis of kdd dataset attributes - class wise for intrusion detection, Procedia Computer Science 57 (2015) 842–851, 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015). doi:https://doi.org/10.1016/j.procs.2015.07.490.
URL https://www.sciencedirect.com/science/article/pii/S1877050915020190

[41] K. Siddique, Z. Akhtar, F. Aslam Khan, Y. Kim, Kdd cup 99 data sets: A perspective on the role of data sets in network intrusion detection research, Computer 52 (2) (2019) 41–51. doi:10.1109/MC.2018.2888764.

[42] D. Protic, Review of kdd cup '99, nsl-kdd and kyoto 2006+ datasets, Vojnotehnicki glasnik 66 (2018) 580–596. doi:10.5937/vojtehg66-16670.

[43] D. Zheng, Z. Hong, N. Wang, P. Chen, An improved lda-based elm classification for intrusion detection algorithm in iot application, Sensors 20 (2020) 1706. doi:10.3390/s20061706.

[44] N. Moustafa, J. Slay, Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set), in: 2015 Military Communications and Information Systems Conference (MilCIS), 2015, pp. 1–6. doi:10.1109/MilCIS.2015.7348942.

[45] N. Moustafa, J. Slay, The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set (2016) 1–14doi:10.1080/19393555.2015.1125974.

[46] R. Fontugne, P. Borgnat, P. Abry, K. Fukuda, Mawilab : Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking, 2010, p. 8. doi:10.1145/1921168.1921179.

[47] B. Brao, K. Swathi, Fast knn classifiers for network intrusion detection system, Indian Journal of Science and Technology 10 (2017) 1–10. doi:10.17485/ijst/2017/v10i14/93690.

[48] W. Hwang, K. Wen, Fast knn classification algorithm based on partial distance search, Electronics Letters 34 (21) (1998) 2062–2063. doi:10.1049/el:19981427.

[49] Y.-L. Qiao, J.-S. Pan, S.-H. Sun, Improved partial distance search for k nearest-neighbor classification, Vol. 2, 2004, pp. 1275 – 1278 Vol.2. doi:10.1109/ICME.2004.1394456.

[50] P. Bringas, I. Santos, Bayesian Networks for Network Intrusion Detection, 2010. doi:10.5772/10069.

[51] Y. Jia, M. Wang, Y. Wang, An network intrusion detection algorithm based on new deep neural network, IET Information Security 13 (08 2018). doi:10.1049/iet-ifs.2018.5258.

[52] M. Sarhan, S. Layeghy, N. Moustafa, M. Portmann, Towards a standard feature set of nids datasets (01 2021).

Appendix A

Generating Datasets through Al Pastor

During this section we will present Al Pastors' interface and the process of creating new datasets.

A.1 Command Line

The various options of Al Pastor are presented bellow:

```
usage: al_pastor.py [-h] -p pcap [-s snort] [--sc snort-config] [-a
    argus]
                [--ac argus-client] [--ds] [--da] [--csv] [-o O]

Process some integers.

optional arguments:
  -h, --help          show this help message and exit
  -p pcap             location of pcap file to parse
  -s snort            location of snort bin
  --sc snort-config   location of snort configuration
  -a argus            location of argus bin
  --ac argus-client   location of argus client bin
  --ds                do not run snort
  --da                do not run argus
  --csv               generate csv files
  -o O                output directory
```

When the *–da* or *–ds* options are used, the associated functionality is omitted from the output (*–da* does not generate a flow file, and *–da* does not label the packets).

A.2 Dataset Generation

The process of creating a dataset is rather straightforward. As seen below, we must specify the pcap file to be used as input as well as the output folder:

```
(thesis) ➜  thesis git:(main) ✗ python al_pastor.py -s snort --sc ../snort_instance/etc/snort.lua -a argus --ac ra --csv -p pcaps/smallFlows.pcap  -o csv
[+] Getting pcap's meta info...
[+] Constructing Netflow...
[+] Using Snort to analyze packets...
WARNING: replace requires inline mode and DAQ with replace capability
[+] Parsing PCAP file:
|                                        | 14261/14261 [100%] in 1:26.5 (164.81/s)
[+] Packets parsed:
eth_ip_tcp[Mal packets/Total packets]: 142/11645

eth_ip_tcp_tls[Mal packets/Total packets]: 77/1449

eth_ip_tcp_tls_tls[Mal packets/Total packets]: 0/614

eth_ip_udp[Mal packets/Total packets]: 61/433

eth_arp[Mal packets/Total packets]: 8/18

eth_ip_udp_dns[Mal packets/Total packets]: 0/68

eth_ip_icmp[Mal packets/Total packets]: 34/34

[+] Generating CSV files:
[+] Cleaning up
```

Al Pastor was able to extract the relevant dataset information from the file *pcaps/smallFlows.pcap* in this case. The amount of risks found in the packets is shown next to each dataset displayed on the interface (also on the stats.txt file produced).

The resulting datasets were placed in the folder csv (since we specified this to be the output folder via *-o csv*). The folder now contains the protocol header datasets (*_traffic.csv files) as well as the Packet Flow dataset (flow.csv).

```
(thesis) ➜  thesis git:(main) ✗ ls -1 csv/smallFlows
eth_arp_traffic.csv
eth_ip_icmp_traffic.csv
eth_ip_tcp_tls_tls_traffic.csv
eth_ip_tcp_tls_traffic.csv
eth_ip_tcp_traffic.csv
eth_ip_udp_dns_traffic.csv
eth_ip_udp_traffic.csv
flow.csv
stats.txt
```

Appendix B

Analysing Protocol Header Datasets through Tensorflow

The next sections will demonstrate how to work with the Al Pastor Protocol dataset. We will be using the Neural Network Classes included in the Tensorflow library.

To begin, we import the required libraries.

```
import tensorflow as tf
from tensorflow.keras.utils import plot_model
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from google.colab import drive
from sklearn.preprocessing import OneHotEncoder
from sklearn.utils.class_weight import compute_class_weight
from sklearn.utils import shuffle

# label category for noise
NOISE_CATEGORY = 5
# percentage of noise entries in the dataset
NOISE_RATIO = 0.02
RANDOM_SEED = 4931
```

Following that, we attempt to read the data from the associated.csv file. The data is partitioned into input/output data frames, and the unique labels are retrieved (a new unique label called 'Noise' is added to the list of unique labels).

```
dataset=pd.read_csv(csv_file_location)
input = dataset.drop(["severity"], axis=1)
output = dataset["severity"].values.reshape(-1,1)
headers = list(input.columns.values)
categories = np.insert(np.unique(output), 0, NOISE_CATEGORY)
categories.sort()
```

The noise function adds "noisy" packets in the dataset

```
def add_noise(input, output):
  # add random noise data to training
  noise_samples = math.floor(NOISE_RATIO*len(input))
  rdx_idx = np.random.randint(
    0,
    high=len(input),
    size=noise_samples
  )
  noise_input = []
  noise_output = []
  for i in rdx_idx:
    noise_input.append(np.random
        .random_sample((len(input[0]),)))
    noise_output.append(np.array([NOISE_CATEGORY]))
  ohe.fit(noise_output)
  noise_output = ohe.transform(noise_output)
  input = np.concatenate((input, np.array(noise_input)))
  output = np.concatenate((output, np.array(noise_output)))
  return shuffle(input, output)
```

The input and output datasets are preprocessed and divided into training and testing sets. Noise is also added into the samples.

```
ohe = OneHotEncoder(
    categories=[categories], handle_unknown="ignore",
    sparse=False
)
ohe.fit(output)
output_ohe = ohe.transform(output)
# Uncomment the str_headers to parse netflow data
str_headers = []#["Proto", "State", "Flgs", "TcpOpt",]
non_str_headers = [
    hdr for hdr in headers if hdr not in str_headers
```

```
]

ct = make_column_transformer(
    (OneHotEncoder(handle_unknown="ignore"), str_headers),
    (MinMaxScaler(), non_str_headers),
)
ct.fit(input)
input_train, input_test, output_train, output_test =
    train_test_split(
        input, output_ohe, test_size=0.2,
        random_state=42
    )

input_train_n = ct.transform(input_train)
input_test_n = ct.transform(input_test)
input_train_n, output_train = add_noise(
    input_train_n, output_train
)
input_test_n, output_test = add_noise(
    input_test_n, output_test
)
df = pd.Series(np.array([
    x[0] for x in
        ohe.inverse_transform(output_train)
    ]))
```

We can pass Keras weights for each class. Examples from an under-represented class will be given extra attention because of this.

```
cw = compute_class_weight(
    classes=categories, y=df,
    class_weight="balanced"
)
cwd = { idx: cw[idx] for idx in range(len(categories))}
```

Finally we create our model and fit our training data:

```
tf.random.set_seed(RANDOM_SEED)
model = tf.keras.Sequential([
  tf.keras.layers.Dense(60, activation="relu"),
  tf.keras.layers.Dense(40, activation="relu"),
  tf.keras.layers.Dense(
    len(categories), activation="softmax"
  ),
])

```

```
val_cb = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy', patience=20,
    mode='max', restore_best_weights=True
)
model.compile(
  loss = tf.keras.losses.CategoricalCrossentropy(),
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.005),
  metrics=["accuracy"])
model.fit(
    input_train_n, output_train, epochs=40, batch_size=64,
    validation_split=0.4, callbacks=[val_cb],
    class_weight=cwd)
```

We can display a confusion matrix with the results by executing the following:

```
prediction = model.predict(input_test_n)
y_pred = ohe.inverse_transform(prediction).flatten()
y_true = ohe.inverse_transform(output_test).flatten()

cm = confusion_matrix(y_true, tf.round(y_pred))
test_categories = np.unique(y_pred)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=[
        "type:{}".format(x)
        for x in test_categories
    ]
)
disp.plot()
plt.show()
```