Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, ΕΛ.ΜΕ.ΠΑ. Κρήτης

Ελληνικό Μεσογειακό Πανεπιστήμιο

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πρόγραμμα Σπουδών Μηχανικών Πληροφορικής ΤΕ

Τίτλος:

Σύστημα ανίχνευσης και πρόληψης εισβολών με βάση τη μηχανική μάθηση σε σχεδόν πραγματικό χρόνο με χρήση eBPF.

Title:

Machine Learning-based near real time Intrusion Detection and Prevention System using eBPF.

Κωστόπουλος Σταμάτιος (ΤΠ4861)

Επιβλέπων εκπαιδευτικός : Μαρκάκης Ευάγγελος

Επιτροπή Αξιολόγησης :

- **Μαρκάκης Ευάγγελος**
- **Μαστοράκης Γεώργιος**
- **Παναγιωτάκης Σπυρίδων**

Ημερομηνία παρουσίασης: 25/08/2023

Ευχαριστίες

Πρώτα απ' όλα, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Δρ Ευάγγελο Μαρκάκη, Υπεύθυνο του Εργαστηρίου Pasiphae και Επίκουρο Καθηγητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ελληνικού Μεσογειακού Πανεπιστημίου, για την καθοδήγηση, τις παρατηρήσεις και τη βοήθειά του στην ολοκλήρωση της πτυχιακής εργασίας μου.
Επιπλέον, θα ήθελα να ευχαριστήσω τον επιστημονικό συνεργάτη του Εργαστηρίου Pasiphae, Νικόλαο Αστυρακάκη για τις συμβουλές που μου παρείχε κατά τη διάρκεια υλοποίησης της πτυχιακής μου εργασίας, και για τις πολύτιμες διορθώσεις του.
Επιπλέον, ένα θερμό ευχαριστώ στην οικογένεια μου, τους φίλους μου και την κοπέλα μου για την αμέριστη συμπαράστασή τους κατά τη διάρκεια των σπουδών μου.

Abstract

As technology evolves rapidly, more and more critical infrastructures are going online. Malicious individuals are trying to exploit such infrastructures, thus cyber-attacks have become a major issue for users and businesses. Various network security software applications are developed to prevent or mitigate cyber-attacks; however, with a low success rate [1], as more than three billion zero-day [2] attacks were reported in a calendar year in the USA and Australia according to Symantec Internet Security Threat Report[1]. Current software applications struggle to confront the more sophisticated malware that cybercriminals use. Additionally, network security software applications, which utilize network packets for detecting cyber-attacks, consume a great amount of power and system resources, such as Random Access Memory (RAM), Disk, Central Processing Unit (CPU), etc.

After researching and reviewing multiple technologies that can be employed to implement optimal security systems, this thesis proposes a cyber-security software application named eIDPs. The proposed solution employs novel technologies for detecting, analyzing, and preventing various network attacks, while utilizing minimum computer resources, namely: the Extended Berkeley Packet Filter (eBPF), which can run virtualized functions directly in the kernel, and Machine Learning (ML) for detecting, analyzing, and preventing various network attacks, while utilizing minimum computer resources.

The use of novel technologies resulted in a better, efficient attack detection and prevention system compared to the current state-of-art network intrusion detection and prevention systems, such as Snort[2]. A comparison was conducted between the solution proposed in this thesis and the Snort software, in a closed test environment. Slight modifications were performed on the Snort detection schema for utilizing the same ML model internally, in order to perform equal measurements between the proposed solution and the Snort software. The evaluation results showcased that eIDPS are vastly more lightweight and efficient in detecting and preventing malicious activities.

[1] https://docs.broadcom.com/doc/istr-03-jan-en
[2] https://www.snort.org/

Περίληψη

Καθώς η τεχνολογία εξελίσσεται ραγδαία, όλο και περισσότερες κρίσιμες υποδομές συνδέονται στο διαδίκτυο. Κακόβουλα άτομα προσπαθούν να εκμεταλλευτούν αυτές τις υποδομές, με αποτέλεσμα οι επιθέσεις στον κυβερνοχώρο να έχουν γίνει μείζον ζήτημα για τους χρήστες και τις επιχειρήσεις. Διάφορες εφαρμογές λογισμικού ασφάλειας δικτύων αναπτύσσονται για την πρόληψη ή τον μετριασμό των κυβερνοεπιθέσεων- ωστόσο, με χαμηλό ποσοστό επιτυχίας [1], καθώς περισσότερες από τρία δισεκατομμύρια επιθέσεις zero day [2] αναφέρθηκαν σε ένα ημερολογιακό έτος στις ΗΠΑ και την Αυστραλία σύμφωνα με την έκθεση Symantec Internet Security Threat Report. Οι τρέχουσες εφαρμογές λογισμικού δυσκολεύονται να αντιμετωπίσουν το ολοένα εξελισσόμενο κακόβουλο λογισμικό που χρησιμοποιούν οι εγκληματίες του κυβερνοχώρου. Επιπλέον, οι εφαρμογές λογισμικού ασφάλειας δικτύου, οι οποίες χρησιμοποιούν πακέτα δικτύου για τον εντοπισμό κυβερνοεπιθέσεων, καταναλώνουν μεγάλη ποσότητα ενέργειας και πόρων συστήματος, όπως μνήμη τυχαίας προσπέλασης (RAM), δίσκος, κεντρική μονάδα επεξεργασίας (CPU) κ.λπ.

Μετά από έρευνα και εξέταση πολλαπλών τεχνολογιών που μπορούν να χρησιμοποιηθούν για την εφαρμογή βέλτιστων συστημάτων ασφαλείας, η παρούσα πτυχιακή εργασία προτείνει μια εφαρμογή λογισμικού κυβερνοασφάλειας με την ονομασία eIDPs. Η προτεινόμενη λύση χρησιμοποιεί νέες τεχνολογίες για την ανίχνευση, την ανάλυση, και την αποτροπή διαφόρων επιθέσεων δικτύου, με ταυτόχρονη χρήση ελάχιστων υπολογιστικών πόρων, και συγκεκριμένα: το Extended Berkeley Packet Filter (eBPF), το οποίο μπορεί να εκτελεί εικονικές λειτουργίες απευθείας στον πυρήνα του συστήματος, και τη Μηχανική Μάθηση (ΜΜ) για την ανίχνευση, την ανάλυση, και την αποτροπή διαφόρων επιθέσεων δικτύου, με ταυτόχρονη χρήση ελάχιστων υπολογιστικών πόρων.

Η χρήση νέων τεχνολογιών οδήγησε σε ένα καλύτερο και αποτελεσματικότερο σύστημα ανίχνευσης και πρόληψης επιθέσεων σε σύγκριση με τα τρέχοντα σύγχρονα συστήματα ανίχνευσης και πρόληψης εισβολών στο δίκτυο, όπως το Snort . Πραγματοποιήθηκε σύγκριση μεταξύ της λύσης που προτείνεται στην παρούσα πτυχιακή εργασία και του λογισμικού Snort, σε κλειστό περιβάλλον δοκιμών. Πραγματοποιήθηκαν μικρές τροποποιήσεις στο σχήμα ανίχνευσης του Snort για τη χρήση του ίδιου μοντέλου ΜΜ στο εσωτερικό του, προκειμένου να πραγματοποιηθούν ισότιμες μετρήσεις μεταξύ της προτεινόμενης λύσης και του λογισμικού Snort. Τα αποτελέσματα της αξιολόγησης έδειξαν ότι το eIDPS είναι πολύ πιο ελαφρύ και αποτελεσματικό στην ανίχνευση και την πρόληψη κακόβουλων δραστηριοτήτων.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Established network security systems

Frequently occurring cyber-attacks have become a major problem for corporations and individuals, according to Albahar M. et al. [3]. Towards securing computer systems and networks, various software applications are developed to mitigate cyber-attacks. There are two main variations of such cybersecurity software: (1) the Network Intrusion Detection Systems (NIDS) [4] and (2) the Network Intrusion Prevention Systems (NIPS) [5]. NIDSs are a sub-category of Intrusion Detection Systems (IDS) that also include Host Intrusion Detection Systems (HIDS) [6].
NIDS and NIPS are systems that supervise a network or a computer system for malicious activities by analyzing the incoming network traffic. The main difference between them is that the former detects attacks and notifies the user that something malicious is happening and the latter detects an attack, notifies the user, and additionally tries to prevent it.
On the other hand, an HIDS acts as a monitor for the operating system, by supervising files, folders, and actions of a computer system for malicious changes. Moreover, a variation of such systems can employ a hybrid detection scheme that can detect malicious activities and anomalies in a network or in a computer file system by combining different strategies and algorithms. Such systems are named Hybrid IDS (H-IDS) [7].
In this thesis, the focus is placed on NIDS and NIPS, rather than HIDS. In Figure 1 below, a concept diagram of a NIDS software in a network is depicted. In this concept, the NIDS system along with a firewall are the "first layer" of security in this architecture. They are both attached to a router to monitor the egress and the ingress traffic of the underlying network.

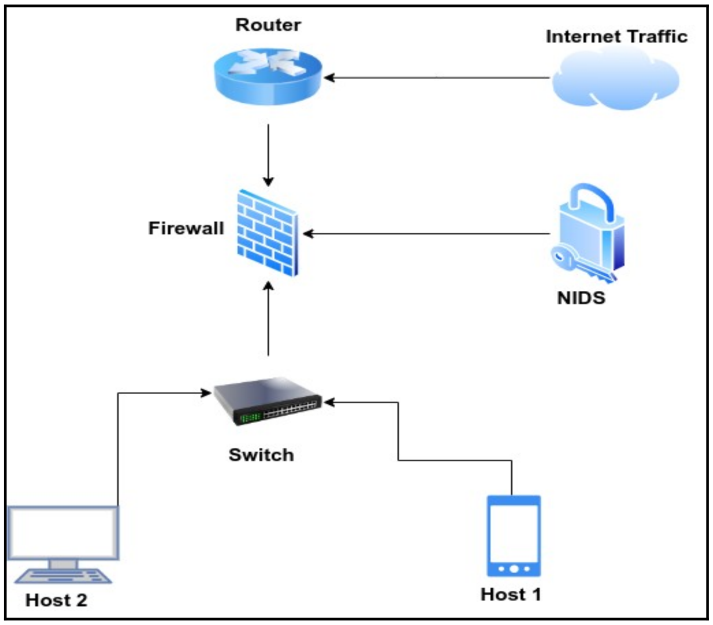


Figure 1 NIDS Coceptual Diagram

Furthermore, in Figure 2, the conceptual diagram of a HIDS is displayed. In addition to the firewall and the NIDS, the HIDS extends security in the computer hosts of the network. The HIDS monitors the hosts and collects events that happen inside the system. If they detect an abnormal event, such as worms [8]or viruses [9], they log and report the findings to the users.

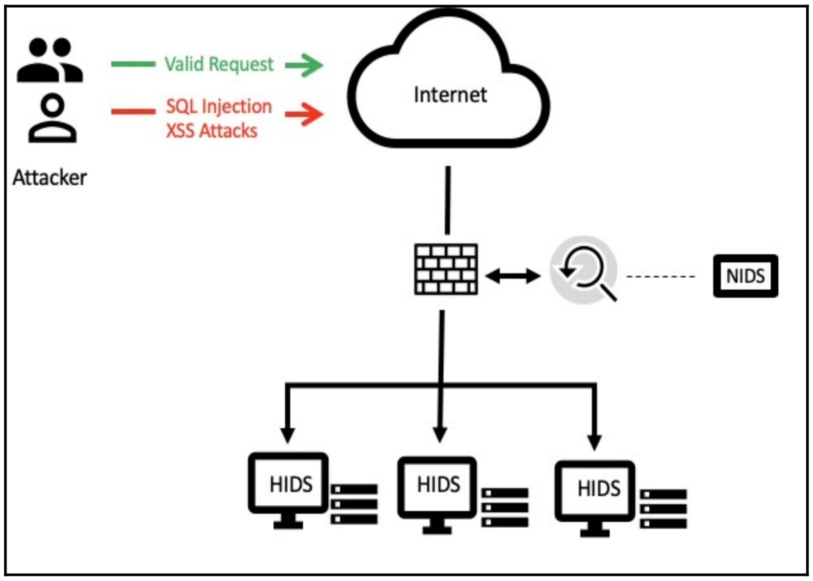


Figure 2 HIDS Conceptual Diagram

In Figure 3, the conceptual diagram of a NIPS is shown. This system works similarly to a NIDS. NIPS is also connected to the network firewall and together with the firewall they are the "frontier defense" in a network.
Moreover, a NIPS is required to be connected to the router as it needs to have direct access to the ingress and egress network traffic. Therefore, when a potentially malicious packet arrives in a network interface of the router, the NIPS analyzes it and, if it matches with a rule that instructs the NIPS to drop it, will prevent it from reaching deeper in the network.

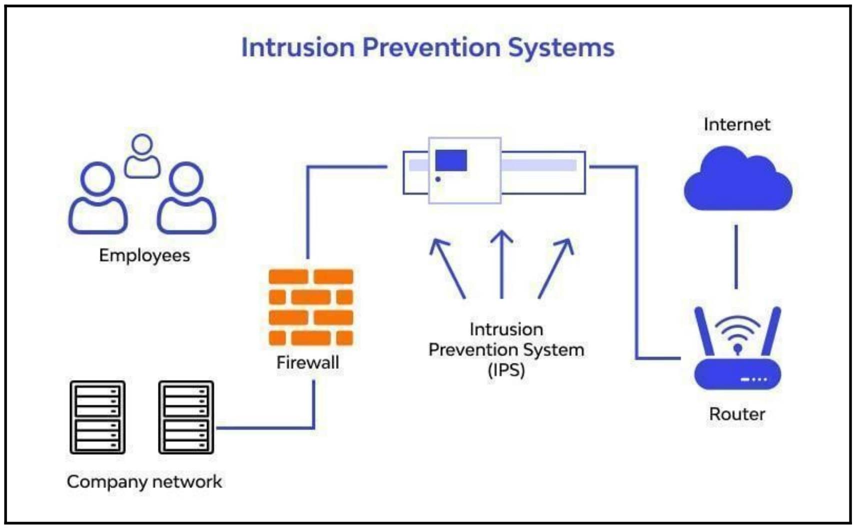


Figure 3 NIPS Conceptual Diagram

In Figure 4, the conceptual diagram of a H-IDS is described. This system combines different strategies for anomaly detections. In this scenario, the system utilizes a signature-based detection scheme together with a Machine Learning [10] anomaly detection scheme [11]. After analyzing the incoming data with both schemes, the system decides whether the incoming traffic is an attack or not and notifies the user.

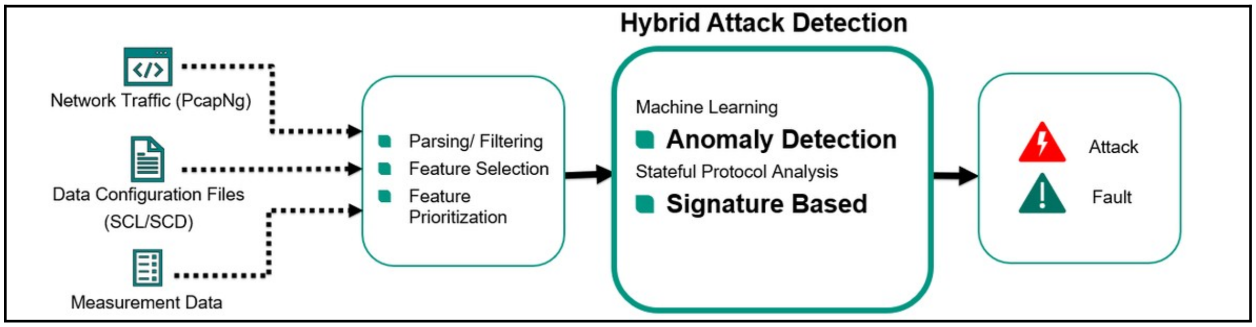


Figure 4 Hybrid-IDS Conceptual Diagram

1.2. Detection Techniques

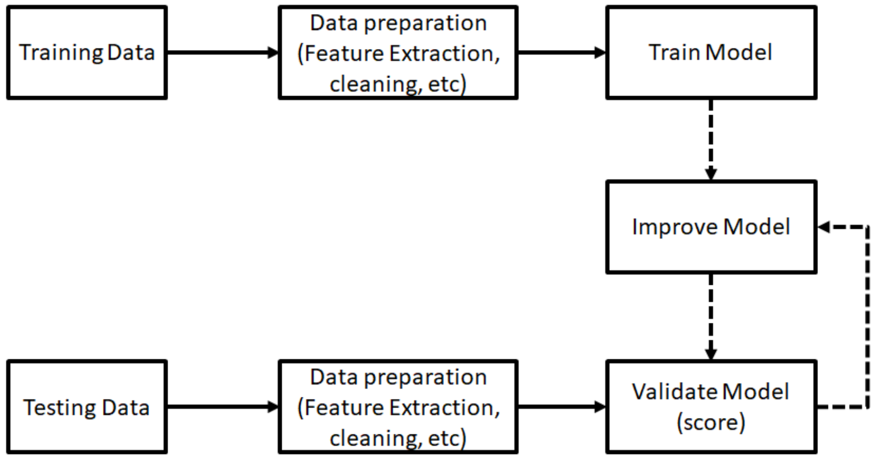
The detection techniques of a NIDS are divided in three main variations: (1) the Signature-based detection technique, which is a method that compares incoming packets with pre-configured attack patterns named “signatures”; (2) the Statistical anomaly-based detection technique, which is a method that compares the incoming traffic with an established dataset that has predetermined benign traffic statistics. For instance, if a packet differs from what the system “thinks” that constitutes normal traffic, it categorizes it as an anomaly/malicious activity; last but not least, (3) the Stateful protocol analysis detection technique that compares the protocol of an incoming packet (e.g., TCP) with pre-determined profiles of an accepted benign activity.
Finally, a NIPS system also uses a detection scheme to detect attacks. In addition to that, the user needs to instruct the NIPS of “what” action needs to be taken, for each kind of detection, according to a specific rule. The incoming packet is matched with a detection signature. Thus, when the system detects an attack with one or more of the previously mentioned detection schemes, the rule that the user created previously is triggered, and it performs the appropriate action, e.g., Packet Drop action.

1.3. Machine Learning (ML)

The efficiency of a NIDS system can be improved by using ML algorithms. The ML technology has a plethora of use cases. The most notable are Big Data [12], the medical field [13], and cybersecurity. Progressively, people are using ML for various cybersecurity purposes, such as data analysis, risk exposure assessment, etc., but also for detecting cyber-attacks, as it offers great and precise results in detecting malicious traffic [14]. A combination of ML with NIDS is easier due to the nature of network traffic analysis. Network traffic involves large volumes of data, making it suitable for ML algorithms to analyze and identify patterns or anomalies, as most of these algorithms need a massive input of sample data to have a valid outcome on detections.

In Figure 5, the basic data flow of an ML algorithm is depicted. At the outset, there is a collection of data, referred to as the dataset, that serves as an input to the model. This data is prepared for training by cleaning unnecessary features (pieces of information about an item inside the dataset). Afterwards, the model gets trained with the dataset using an ML algorithm and gets validated simultaneously. Following, a validation process is required to improve the model’s efficiency by testing data at the same time while

training the model. In addition, the testing data of the validation process sometimes is a different dataset, dedicated to validation, or it can be a fraction of the training dataset.
Additionally, the model's outputs may vary based on the ML algorithm used to train it. Some algorithms are better suited to particular datasets and applications than others. For instance, if a relatively small dataset is used for training an ML model that utilizes a complicated algorithm, then the model can learn and adapt to it too well, resulting in production of false outputs.


Training Data
Data preparation (Feature Extraction, cleaning, etc)
Train Model
Improve Model
Testing Data
Data preparation (Feature Extraction, cleaning, etc)
Validate Model (score)
Figure 5 Simplified Machine Learning Data Flow

1.4. eBPF technology

eBPF [15] is a novel technology on the rise which can run virtualized functions, directly in the kernel. This technology is an extension of the Berkeley Packet Filter (BPF) [16] that was released in 1992, for UNIX [17] type systems and Windows [18]. This technology is primarily used for network traffic analysis and packet filtering. Figure 6 below, depicts an overall eBPF architecture. More specifically, the generic eBPF architecture consists of a program that runs in a user space and gets invoked by the kernel when a certain event happens, called "hook", such as system calls or a network packet arrival.
Additionally, in the below architecture diagram, someone can preview the eBPF program, which can be compiled by a compiler. The most popular compilers for eBPF functions in C-programming language are the Clang[3] and the LLVM[4]. The eBPF software needs to be translated (compiled) into bytecode, in order to be injected into the kernel later.
Since the eBPF resides in the kernel, it poses some limitations for the kernel's security. The kernel shall not be stopped or crashed at any time. Additionally, eBPF has a limit of maximum one million lines of code, requires no unbounded loops, and prohibits memory relocation. To avoid all these issues, eBPF has a verification module that examines each eBPF software and it ensures that there are no errors in the program or violations of the aforementioned limitations. If there are no errors, the eBPF bytecode is inserted in the just-in-time (JIT) compiler [19] for execution, otherwise, the eBPF software gets rejected and a notification is shown to the user, with the indication of "unsafe" application.
Finally, the program in the kernel can communicate with the user space through the eBPF maps. These maps have key-value tables with entries, and they are shared with the user space program.

[3] https://clang.llvm.org/
[4] https://llvm.org/

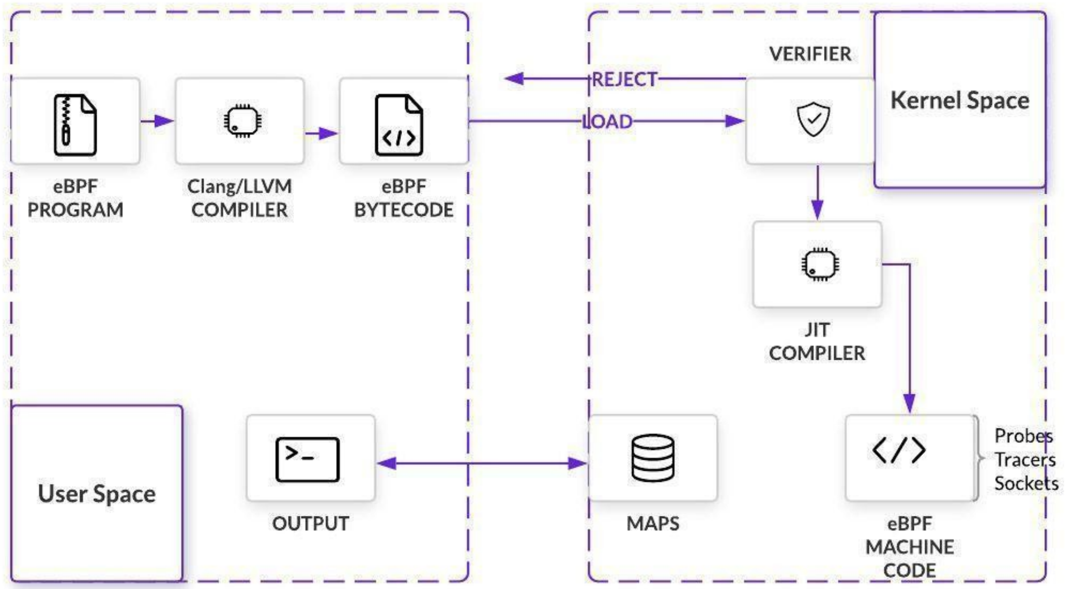


Figure 6 eBPF Architecture

1.5. XDP

The eBPF technology has evolved quickly in recent years. Every Linux kernel release adds support of new features for eBPF. One feature that got released in 2016 is the eXpress Data Path (XDP) [20]. In Figure 7, the XDP's architecture is shown. This diagram presents an application which utilizes the XDP feature, which is basically an eBPF-based program that gets hooked into the Network Interface Card (NIC) driver. In other words, the NIC card can communicate with the program and get instructions on what to do with the network traffic, in near real-time.
Furthermore, there are two buffers called rx_ring and sk_buff. The former is a buffer that lies in the NIC driver and stores pointers to the incoming packets. Consequently, the XDP utilizes it to process packets fast. The latter is a buffer inside the kernel, that stores packets' information like its length and data in the network stack. More particularly, when a packet arrives, before the packet is copied to the sk_buff, it gets on the rx_ring buffer where the XDP program processes it. The XDP program can instruct the NIC driver to various actions with the egress and ingress traffic through certain functions. These functions are the "***XDP_PASS***", which lets a packet through the system, the "***XDP_DROP***" that discards a packet and prevents it from reaching the system, the "***XDP_ABORTED***", which drops a packet and returns an error code, the "***XDP_TX***", which bounces the packet to the same NIC it arrived on and finally, the "***XDP_REDIRECT***", which redirects a packet to a different NIC.

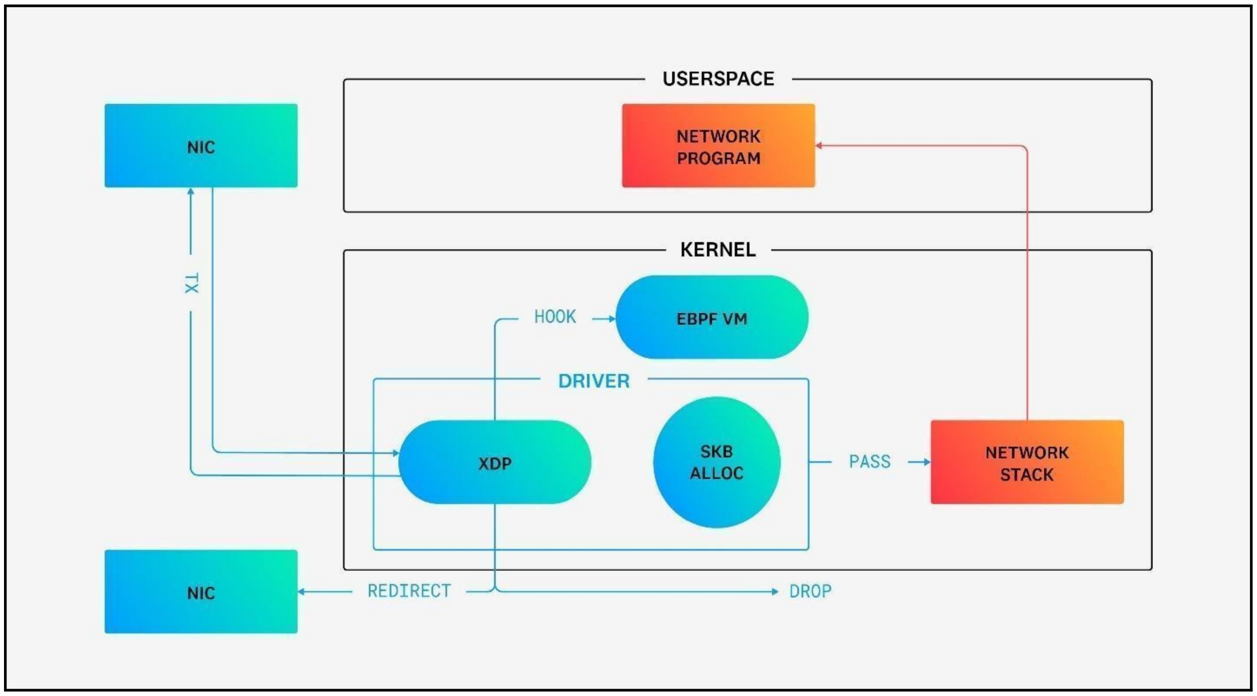


Figure 7 XDP Architecture

1.6. eBPF and ML

eBPF technology can be utilized with the previously mentioned NIDS and NIPS systems alongside ML algorithms, to monitor a computer network and prevent malicious activity. By elevating the aforementioned technologies, packet processing can happen fast and vastly, based on experiments from literature .More specifically, according to [21], eBPF and XDP applications have notable differences in using system resources, in contradiction to other similar applications. That is because XDP scales the CPU usage with the packet load that is facing, instead of dedicating CPU cores exclusively to packet processing. This means that less CPU and RAM are required. Moreover, because eBPF operates within the kernel and offloads traffic analysis to the NIC, packet processing occurs significantly faster than with security software that processes packets in the operating system's user space.

1.7. Thesis Contribution and Proposed System (eIDPS)

To the best of our knowledge, current software applications underperform regarding detection and prevention of suspicious and probably malicious actions in a computer system simultaneously with the aid of ML, , due to the fact that they need an enormous CPU power to make the packet processing and execute the preventions rules simultaneously. Thus, this thesis proposes the eBPF Intrusion Detection and Prevention System (hereinafter “eIDPS”) that is a virtualized kernel solution utilizing eBPF and XDP, combined with a sophisticated ML algorithm, which can detect and prevent malicious packets in near real-time. The proposed solution is evaluated by performing various measurements on performance, efficiency, and capabilities in a closed test environment.

1.8. Thesis Structure

There are seven chapters including the introduction above. The document is structured as follows: the subsequent chapter is the “State-of-the-art” chapter, which includes related literature work conducted in recent years. Next, the “Technology Enablers” chapter is presented, which outlines the tools and software that is used to build and evaluate the application. The “Evaluation” chapter presents the experiments that were conducted to evaluate this thesis’ proposed model and compare the resulting solution with similar applications. Finally, the “Conclusion and Discussion” chapter summarizes the results of the evaluation and discusses this thesis’ future work directions.

2. State of the Art

2.1. Machine Learning in Network Intrusion Detection systems.

NIDSs are among the most popular systems for detecting cyber-attacks C. Xu and J. Shen in [22], developed a NIDS using a Deep Neural Network algorithm (DNN), an ML algorithm that gets incorporated into the system and gets trained for detecting cyber-attacks. DNN was combined with Gate Recurrent Units (GRU), which is a simpler Long Short-Term Memory (LSTM), Multilayer Perceptron (MLP), and a SoftMax module, which takes an input vector of arbitrary real numbers and transforms them into probability distributions, where each element represents the likelihood of a corresponding outcome. Furthermore, the authors used the KDD-99 and NSL-KDD datasets to train their ML models. Experiments using these datasets demonstrated the system's substantial efficiency. The overall detection rates for KDD-99 and NSL-KDD were 99.42% and 99.31%, with false positive rates of 0.05% and 0.84%, respectively. On the other hand, they used LSTM instead of GRU and comparative studies were conducted. GRUs alongside an MLP performed better than LSTM, however, the system suggested in this article primarily depends on theoretical validation and its practical application remains to be confirmed.
Moreover, there are several attacks recorded against a network or system. On the network system, there is a chance that cyber-attacks such as "black holes," "gray holes," and "wormholes" may happen. These attacks aim to alter data present on any system or to steal information from it. Attacks on the system include DoS, probe, snort, r2l, and other methods to misuse the data. Hence, NIDSs have been introduced to protect the system from such threats. In [23], researchers suggest that NIDS can be enhanced if it is used with principal component analysis (PCA), which is a technique for analyzing big datasets using the Random Forest (RF) algorithm. The suggested method effectively handles the detection of network intrusions compared to previously used algorithms such as Support vector machine (SVM), Naive Bayes, and Decision Tree. The suggested approach has an error rate of 21% compared to 2.67%, 3.49%, 0.78%, for SVM, Naïve Bayes, and Decision Tree, respectively. Moreover, its accuracy rate is 96.78%, in comparison with the corresponding 84.34%, 80.85%, and 89.91%, rates. The suggested approach has the potential to considerably increase detection rates and false error rates.
According to [24], due to the enormous number and constant change and evolution of cyber-attacks techniques the datasets that are used for training ML-based NIDSs should be routinely updated and benchmarked. A DNN was investigated to create a flexible and powerful IDS that can identify and categorize unanticipated and unpredictable cyber-attacks. In both HIDS and NIDS, the proposed architecture outperformed traditional ML classifiers currently in use. However, ML classifiers are only partially capable of identifying cyber-attacks in the KDDCup-99 dataset's attacks' category. None of the ML classifiers could increase the cyber-attacks detection rate using this dataset.

2.2. eBPF and XDP

Originally, computer networks incorporated the use of communication protocols in the hardware of network devices, making it challenging to update/change them to comply with the demands of modern times. In [25], researchers presented eBPF and XDP, the limitations that they have and how they are used in packet processing procedures. Consequently, the growth of latest research projects using eBPF as well as XDP, shows that these technologies have a lot of potential. Moreover, this technology can be used together as a means of providing new functionalities in the data plane, which is a network component that carries user traffic, and more specifically, through the data planes. These functionalities can protect the user's system from potential cyber-attacks. Finally, eBPF and XDP can be used in the development of novel research prototypes and network solutions.
Literature showed that an eBPF program combined with ML algorithms is faster at processing packets than a user-space program, using the same algorithms. In accordance with M. Bachl in [26], it is feasible

to create a NIDS using ML-based algorithms, such as Decision Trees, combined with eBPF, where the Decision Tree decides if a packet is malicious and needs to notify the user accordingly. After training their proposed model using the CIC-IDS-2017 dataset, an accuracy of 99% was achieved. As a result, the eBPF program showcased a 24% increase in performance compared to the same NIDS and training model in a user space program. However, there are few limitations to the eBPF, such as the floating points. Therefore, the Decision Tree must use fixed-point arithmetic of 64-bit integers. To summarize, this study doesn't provide enough information neither on "how" the comparison was conducted nor provides a table with the overall results.
Moreover, software-based IDSs are slower and analyze fewer packets in a specific amount of time than hardware-based systems. Wang S and Chang J in [27] proposed another NIDS that utilizes eBPF. Their solution contains two programs, one that is running in the kernel through eBPF and one that is running in the user space. The former uses a modified Snort ruleset combined with eBPF to filter the packets while the latter uses Snort for packet filtering. The results show that, after some testing conditions, the eBPF program that was running in the kernel not only offers higher network throughput but also requires less CPU usage. This happens because Snort in the user space uses one thread, thus the single core usage is 100%. Additionally, packet loss happens for the same reason as bottlenecks occur when the traffic load reaches over 200mbps.
According to L. Caviglione [28], attackers are continually developing novel strategies to evade signature- and rule-based detection techniques, making modern malware harder to detect. By using eBPF, software operations can be efficiently tracked and observed in near real-time. To demonstrate the adaptability of the method, the researchers investigated two real-world use case scenarios that employ various attack strategies, namely two processes cooperating by changing the file system and converting network connection attempts buried within IPv6 [29] traffic flows. The findings demonstrate that even basic eBPF algorithms may deliver pertinent information for anomaly identification with low overhead. Furthermore, the adaptability of developing and running such programs enables the extraction of pertinent aspects that may be used to produce datasets for "feeding" AI-powered security frameworks.
In addition to that, eBPF has the ability to use the NIC through XDP for monitoring packets directly. Therefore, D. Scholz et al. in [30] presented the complexities of the two main technologies (eBPF and XDP). The authors showcased the utilization of SmartNICs – which are NICs that can get offloaded tasks from CPU – in packet processing to create a processing pipeline that is more effective. Finally, they offered specific details on how the aforementioned technologies can be used to mitigate Distributed Denial of Service (DDoS) attacks. They found that the most effective method reduces CPU consumption and dropping rates reduced by combining hardware filtering on the SmartNIC with XDP software filtering on the host. Operating a portion of the filtering pipeline on the SmartNIC, without XDP filtering, the CPU would result in performance degradation and a reduced ability to withstand powerful DDoS attacks. However, according to their research, current SmartNICs can assist in reducing the network load on overloaded servers, but they might not be a complete panacea.
Moreover, in [31] T. Hoiland-Jorgensen demonstrate how XDP can process up to 24 million packets/sec on a single core and highlight the programming model's adaptability with the help of three example use cases: (1) the layer-3 routing case, (2) the inline DDoS protection case and (3) the layer-4 load balancing case. In accordance with their analysis, XDP can handle raw packets at speeds of up to 24 Mbps using just one CPU core. Although they explain that XDP offers additional attractive features that make up for the speed difference, they acknowledge that this is not on par with state-of-the-art kernel bypass-based alternatives.
Furthermore, the issue of performance was addressed in [32] by Oliver Hohlfeld et al. They brought up the network stacks, which are the protocols that the network uses and are challenged by high packet rates of 10 GBit/s or higher. The authors elucidate the advantages and drawbacks of eBPF/XDP-based offloading from user space to either the kernel or a SmartNIC with Virtual Machine (VM) virtualization. In the end, they demonstrated that offloading could improve packet processing, but only if the workload is manageable and tailored to the intended environment. Additionally, their SmartNIC is often

overwhelmed by excessively demanding duties and updating offloaded data, which could be an expensive task.
Based on the results of the literature presented above, and to the best of our knowledge, there is currently no NIDS that can act as a NIPS simultaneously and in near real-time, utilizing ML for detecting anomalies. Most papers suggest NIDSs that can detect malicious packets using ML or Deep Learning (DL) [33] algorithms, but neither prevent nor process them before they reach deeper in the operating system. On the contrary, there are papers showing that the use of eBPF for monitoring computer systems and filtering packets, inside the kernel and before they reach the system, can be beneficial for users since it consumes less system resources. Moreover, relevant literature discusses that eBPF can be used through XDP to offload the analysis of the packets directly to a NIC, consequently making the decision of dropping a packet faster, thus creating a more efficient security system. However, these features are one-dimensional. Adding an ML-based algorithm can expand the capabilities of eBPF as it can provide the system with knowledge of malicious packets, therefore making the detection and prevention quicker and more efficient.

3. Technology Enablers

3.1. Extended Berkeley Packet Filter (eBPF)

eBPF[5] is a technology that can run sand-boxed programs inside the Linux kernel. These applications are isolated from the core of the kernel and cannot interfere with other functions. Moreover, eBPF is used to expand the kernel's capabilities without interfering with the source code or the load of kernel modules. Using a JIT compiler and a verification engine, the operating system ensures that sandboxed applications have similar security levels and the computation performance specifications, as a natively compiled application – embedded functionality of the kernel. eBPF has various use cases such as tracing and monitoring computer systems. In this scenario it is used for networking security through eXpress Datapath, which is an eBPF extension that handles the ingress and egress network traffic.

3.2. eXpress Data Path (XDP)

XDP[6] is a feature of eBPF that is used for networking. It offers fast packet processing as it analyzes the packets in the kernel and the NIC since it is possible to offload network traffic to the NIC. XDP performs actions to the ingress and egress packets, with the functions **"XDP_DROP"**, **"XDP_PASS"**, **"XDP_REDIRECT"** and **"XDP_REJECT"**.

3.3. Neural Networks

An artificial Neural Network is a computer system that mimics the biological neural networks of a human or an animal, according to [34]. The aforementioned system is based on an assortment of nodes called neurons and neurons that are connected together in a linear way are called layers. A neural network has two layers: the input and the output layer. The former acts as an input and the latter as an output. In addition, it may have layers between the input and output layers, called hidden layers. Neurons are similar to synapses in the human brain. Each link can send a signal to one or many neighboring neurons after processing it. This signal can be translated as a real number. Each neuron's output is determined by a nonlinear function that adds up all its inputs. The connections between neurons are called edges. Moreover, the parameter that connects the two basic units in an artificial Neural Network is called weight. The weight of neurons and edges often changes as learning progresses. Such neural networks are trained to learn by analyzing samples. Each sample has known inputs and outputs / results, thus creating probability-weighted correlations between the input and output.

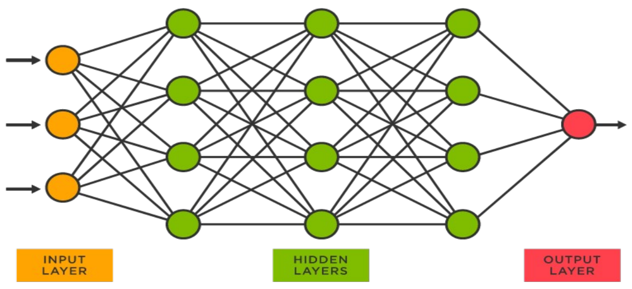


Figure 8: Neural network Conceptual Diagram

[5] https://ebpf.io/
[6] https://www.iovisor.org/technology/xdp

3.4. NSL-KDD Dataset

The NSL-KDD Dataset[7] is often used in ML-based NIDSs as a training and test dataset. It is an updated version of the popular KDD cup99' Dataset[8], solving some of the problems that KDD'99 had, such as outdated attacks, and duplicate entries, according to Tavallaee [35]. NSL-KDD contains approximately 12500 unique entries; thus, each entry is a different packet. Moreover, the entries are classified as anomaly or benign packets. In this thesis, NSL-KDD is used for the training of the ML model included in the proposed solution.

3.5. C-programming language

C-programming language[9] is a versatile and efficient programming language. It is well-known for its low-level capabilities and direct hardware access and is frequently used for system programming. In this thesis, it was used to create the eBPF and XDP functions and libraries for the proposed eIDPS.

3.6. Python Programming language

Python[10] is another versatile programming language that has a lot of use cases. It emphasizes code readability and offers a broad standard library, making it ideal for various applications. Python's interpreted nature allows rapid development and testing, it also has numerous libraries and frameworks, enabling developers to tackle a wide range of projects, from web development to data analysis and artificial intelligence. In this thesis it was utilized to build the Neural Network through frameworks of Keras and Pandas and plotting the training and validation loss and accuracy.

3.7. Keras

Keras[11] is an open-source framework, which offers a Python interface for building neural networks. Moreover, it provides a simplified interface to build TensorFlow[12] neural networks and has detailed documentation. More specifically, TensorFlow is an open-source deep learning framework for building and training machine learning models.

3.8. Pandas Python Library

For the purpose of manipulating and analyzing data, the Python programming language has a library called Pandas[13]. This includes specific data structures and procedures for working with time series and mathematical tables and it is distributed as an open-source library.

3.9. Virtual Box

Virtual Box[14], developed by Oracle, is a hypervisor that can create and run virtual machines on top of bare-metal machines and operating systems. Utilizing this technology assisted in creating multiple

[7] http://www.di.uniba.it/~andresini/datasets.html
[8] https://archive.ics.uci.edu/dataset/130/kdd+cup+1999+data
[9] https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html
[10] https://www.python.org/
[11] https://keras.io/
[12] https://www.tensorflow.org
[13] https://pandas.pydata.org/
[14] https://www.virtualbox.org/

different virtual machines in this thesis, with various operating systems, to perform the laboratory tests required for validation of the proposed eIDPS.

3.10. Kali Linux

Kali Linux[15] is an operating system (of Linux distribution) designed for cybersecurity professionals, penetration testers, and ethical hackers. In the proposed solution, it was installed in virtual machines, for experimenting with eIDPS functionality. In addition to that, the previously mentioned similar application to eIDPS was also installed in Kali to test its functionality.

3.11. Wireshark

Wireshark[16] is a free and open-source tool for analysis, filtering, and previewing of network packets and traffic. This tool is based on tcpdump[17] (a command-line packet analyzer) and pcap-filter[18] (a filter program). Moreover, this tool was used to monitor traffic from virtual machines and perform various metrics during the experimental phase.

4. Implementation

This chapter refers to the implementation of the previously mentioned technologies to build a software application that is capable of detecting and preventing malicious packets in near real-time. The eIDPs uses a neural network for the classification of the network packets and utilizes the offloading of the packet processing in the NIC through XDP.

4.1. Conceptual Diagram

The conceptual diagram in Figure 9 depicts in an abstract level the basic concept of the proposed solution in layers. The layers are shown from top to bottom, namely: the first layer at the top of the diagram is the user-space, where the training and validation of the proposed ML model happens, the second layer is the

[15] https://www.kali.org/
[16] https://www.wireshark.org/
[17] https://www.tcpdump.org/
[18] https://www.tcpdump.org/manpages/pcap.3pcap.html

kernel, which contains the proposed eBPF based IDS/IPS module. Finally, there is also the Data Link Layer, where lies the NIC card, which is responsible for the ingress and egress traffic.

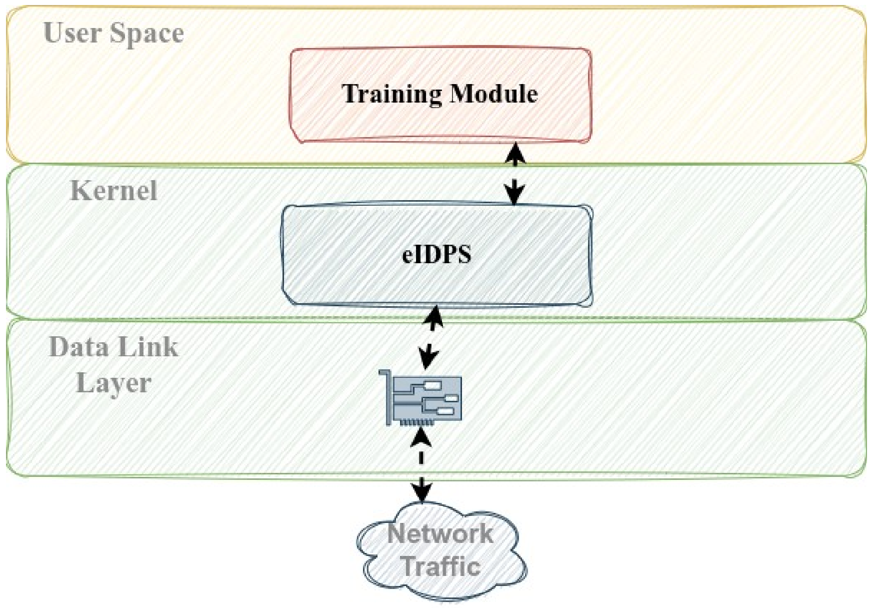


Figure 9 eIDPS Conceptual Diagram

4.2. Architectural Diagram

This section expands on the conceptual diagram. In Figure 10, the Operating System (OS), kernel, and Data Link Layer are depicted in a more detailed way, serving as the foundation of the proposed security solution. In the user space there is a script that compiles the eBPF program and hooks it to the kernel. Moreover, there is a Dataset that acts as an input to the training module, which means that during training the ML model traverses all the entries and tries to learn which packets are malicious and which are benign. The dataset used has a variety of entries of network packets that are either labeled as benign or malicious packets. This results in ML model training with adequate data. Finally, the output of the training module, which is the weights and the bias gets inserted into the kernel.
Furthermore, inside the kernel, there is the eBPF program that functions as a network IDS/IPS module. More specifically, this module contains the Security and Tracing sub-modules. The Security Module accesses the trained model, which is the weights and the bias of the ML model and gives instructions to NIC to drop and/or let a packet go through the system. The Tracing Module is constantly running and checking every packet the NIC receives to see if it is malicious or not. Lastly, the network card in the Data Link Layer plays a pivotal role in the system's security as it is connected to the Tracing Module through XDP. In a nutshell, the NIC handles Network Traffic and, when the eIDPS gives the instruction, it drops a packet without allowing it to reach the kernel. Consequently, packet prevention - intervention - happens in the NIC based on the instruction of the Security module.

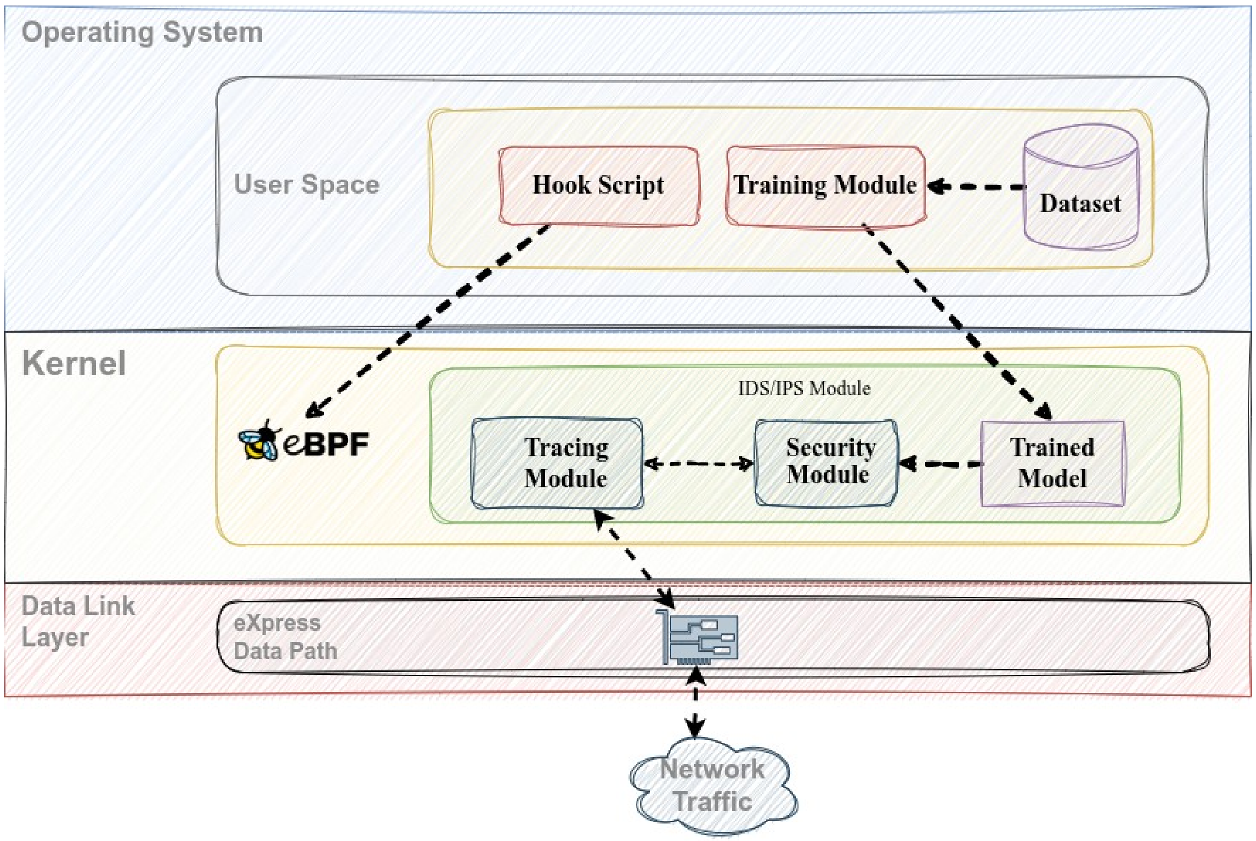


Figure 10 eIDPS Architecture Diagram

4.3. ML Model

This section refers to the implementation of ML model that is part of the solution proposed in this thesis. The NSL-KDD Dataset was selected as the training dataset because it is a simple-class dataset that categorizes packets as anomalies or normal traffic. First, a preprocessing procedure needs to be performed to clean, transform, and prepare the raw data prior to feeding it into the model for training. Afterwards, the newly transformed dataset is fed to the ML model where the training and validation takes place.
More specifically, the NSL-KDD dataset is loaded by using the Pandas library. Then, the column "***class***", which is the dependable value that needs to be predicted, is defined. According to the dataset, the label of column "***class***" is 0 if the packet is benign and 1 if it is malicious. Then the columns that refer to the independent values "***protocol_type***", "***service***", and "***flag***" are dropped, through the function "***df_drop***", which is a function that excludes these columns from the computations. This happens because the values of these columns are defined as string format, so they cannot be computed. Later, the values of the remaining columns of the dataset are normalized, to bring them in a similar range before feeding them to the ML model. Additionally, the dataset has to be split into two sets, namely a training set and a test set, as it is required for later performance evaluation. The split of the dataset is performed by the "***train_test_split***" function with a batch test size of 0.25, which means that 75% of the data are assigned to the training set and 25% of the data will constitute the test set that will be used for later testing. This can be seen in Appendix I. The random state that determines how the data is shuffled prior to the split is set to forty-two (42).
Afterwards, a sequential model is created with the Keras framework. This means that the layers are created in a linear stack. The developed Neural Network has sixty-four (64) neurons with Relu activation, which helps a node to learn complex patterns and it returns binary output, and Adam optimizer, which dynamically adjusts the learning rate for each parameter and it results in faster convergence and better

performance, while training neural networks, in the first layer. In addition to that,), in the second layer there is one neuron with sigmoid activation (it maps the input values to a range between 0 and 1).
Finally, the training begins after the configuration of the model ends with a batch size (the number of samples per gradient update) of 250 and 100 epochs (number of iterations that are happening in the Dataset). Immediately upon the end of training the weights get extracted and inserted into an array along with the bias, thus it is easy to be extracted and inserted into the eBPF program.

4.4. eBPF Program

In this section the eBPF program initialization and startup is described. The eBPF starts with two BPF map type arrays. These arrays keep track of packets that were dropped or passed into the system. Additionally, the weights of the ML model are extracted alongside the bias, and they are both inserted as an array, as it is easier to do mathematical computations with arrays. In the end, two pointers are initialized to assist in keeping track of the beginning and the end of a packet.
In addition to the above, the program receives the incoming network packet and extracts the Ethernet header through the function "***parse_eth***". In Figure 10, the Ethernet packet frames' headers are shown. The header that the program extracts is the Data header, as it contains the information that the program needs to analyze.

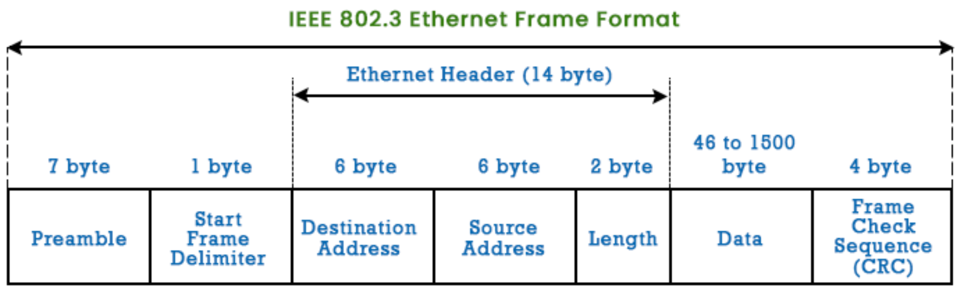


Figure 11 Ethernet Packet headers

Afterwards, the eBPF program checks if the content (data) of the packet is less than the length of the packet, as defined in the headers of the packet. If the result after differentiation is greater than the length, then it allows the packet through to the system, so it will be considered as a non-valid packet i.e., corrupted, incomplete, or improperly formatted. Moreover, this happens to protect the program from running in an infinite loop, thus crashing the kernel. After that, the eBPF program extracts the IPv4 header from the Data header extracted before, through the "***parse_ipv4***" function, to recognize if it is an IP packet. If it is not, it lets it through so the system will oversee it. In Figure 11, the IP packet headers are shown along with their length in bytes. More details are provided in Appendix I.

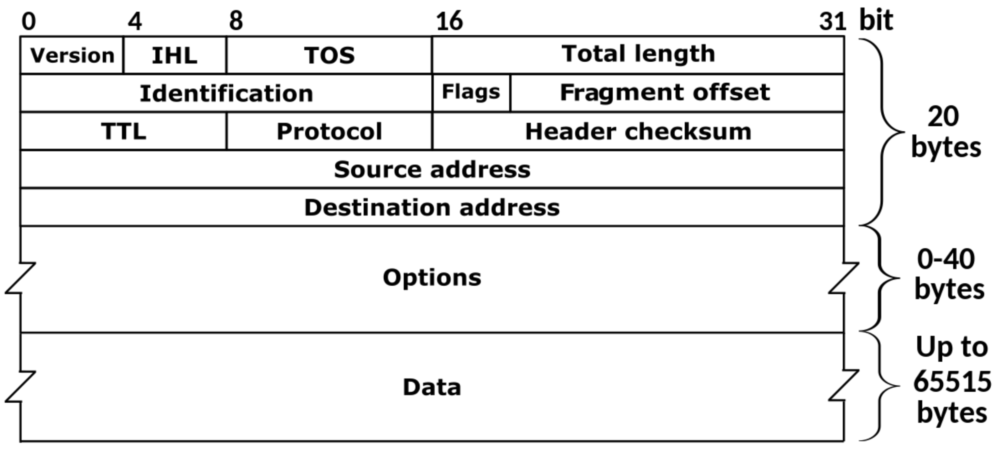


Figure 12 IP Packet Headers

If the header is an IP protocol[19] packet, it goes through a bounded loop that starts iterating the packet headers with the help of the previously mentioned pointers. It begins with the 4th byte until the 54th byte, as the first three bytes are not that important in detecting malicious activity as they contain headers like VoIP, etc. Identification, which is the header in the 4th byte, helps identifying the IP diagrams aiding in detecting malicious activity, thus it is important in identifying if the packet is malicious. Next, inside the loop it multiplies the packets data with the weights and adds the bias $\left(y = bias + weight * x\left[i\right]\right)$, as resulted through consulting paper in $\left[9\right]$. The result stemming from the computation is the Maliciousness Indication Number (MIN). If MIN is greater than 0 it drops the packet through "***XDP_DROP***", while in any other case it lets it go through in the system as the packet is defined as benign, through "***XDP_PASS***" function.

4.5. Compile/Attach script

This section analyzes the script module that hooks the eBPF script inside the kernel to the NIC driver. The bash script that is used compiles the program using Clang and LLVM and hooks it to the kernel using a network interface of the user's choice.
More specifically, it uses the command "***-g -c -02 -S***" to compile the program as follows: (-g) enables the compiler to generate debugging information; (-c) instructs the compiler only to compile the program without linking; (-02) means that it has an optimization level 2, thus better performance when running; (-S) instructs the compiler to make an assembly file and not an object file, which happens because the proposed solution works in a very low level field. Also, it uses the "***-emit-llvm xdp.c***" command that instructs Clang to emit the LLVM IR code (which is a code that sits between the C code and the Machine code).
Afterwards, the command "***-o -|clang_llc -march=bpf -filetype=obj -0 xdp.o***" is used. This command invokes the LLVM backend compiler and specifies the bpf architecture as well as the output object file.
Lastly, the program is hooked to the network interface using the "***ip link set dev $1 xdp obj xdp.o***" command.

4.6. The Remove Script

This section describes the purpose of the second script, the "**remove.sh**" script. Firstly, this script is used to detach the XDP program from the given network interface. The script begins by checking the number,

[19] https://www.impactcybertrust.org/dataset_view?idDataset=945

of interface names that the user has provided. If the number of interfaces is not 1, it displays a message indicating how to use the script and then it exits with message "***Usage: ./compile_attach.sh <interface>***".
If the number of interfaces is 1, the XDP program gets removed from the kernel by using the internal command "***sudo ip link set dev $1 xdp off***".
Later, the script checks the exit status of the previous command, using "***$? -eq 0"***. If the exit status is 0, it means the XDP program was successfully detached from the specified interface and prints a message stating "***XDP Prog removed from <interface>"*** where ***"<interface>***" is the name of the network interface the user provided before.
Following, a flow chart and a sequence diagram are provided for laying a better understanding of the procedures followed by the proposed eIDPS solution.

4.7. Flowchart Diagrams

The proposed solution can be divided into two workflow scenarios. The first one, as depicted in Figure 13, shows the flow of the pre-configuration of the proposed solution. More specifically, the flow begins with the insertion of the NSL-KDD Dataset as an input into the neural network model which serves as the ML algorithm in the proposed solution. Afterwards, the dataset is split and used to train and validate the model. Later, the hook script is executed, which compiles and attaches the program to the kernel. At the same time, the output of the neural network, which is the weights and the bias, is inserted into the network IDS/IPS module, which is found inside the kernel.

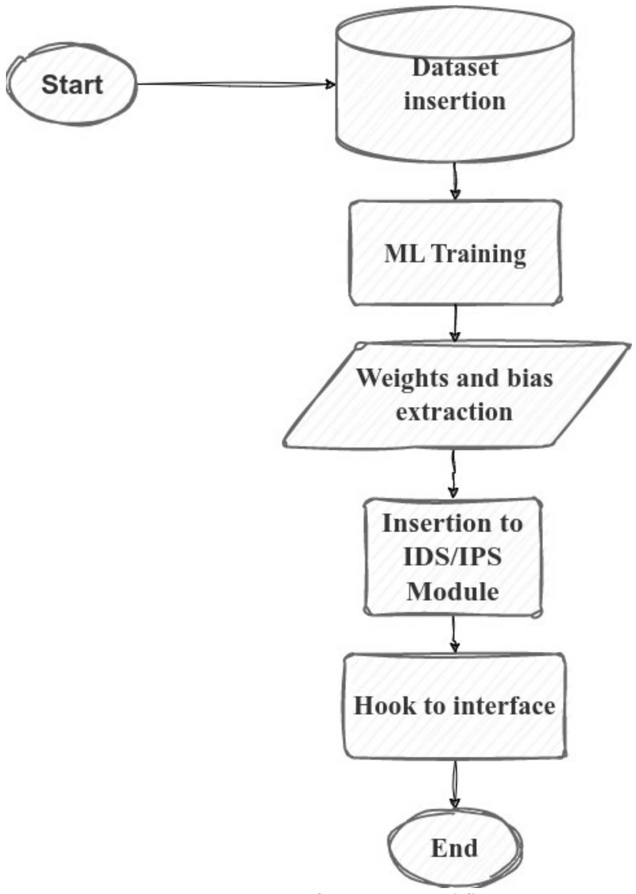


Figure 13 pre-configuration workflow

In the diagram presented in Figure 14, the workflow of the eIDPS when egress or ingress traffic is captured in the NIC is depicted. As can be observed, the internet traffic passes through the NIC card, where all of the incoming packets are processed. After the processing, the retrieval of weights and bias from the eIDPS module, which resides in the kernel, is performed. Afterwards, a computation is made to extract the MIN number in order to determine whether the network packet is malicious or not. If the MIN number is greater than zero (0) then the network packet is classified as malicious and gets dropped, while if it is equal or less than the zero (0) then the packet is let through.

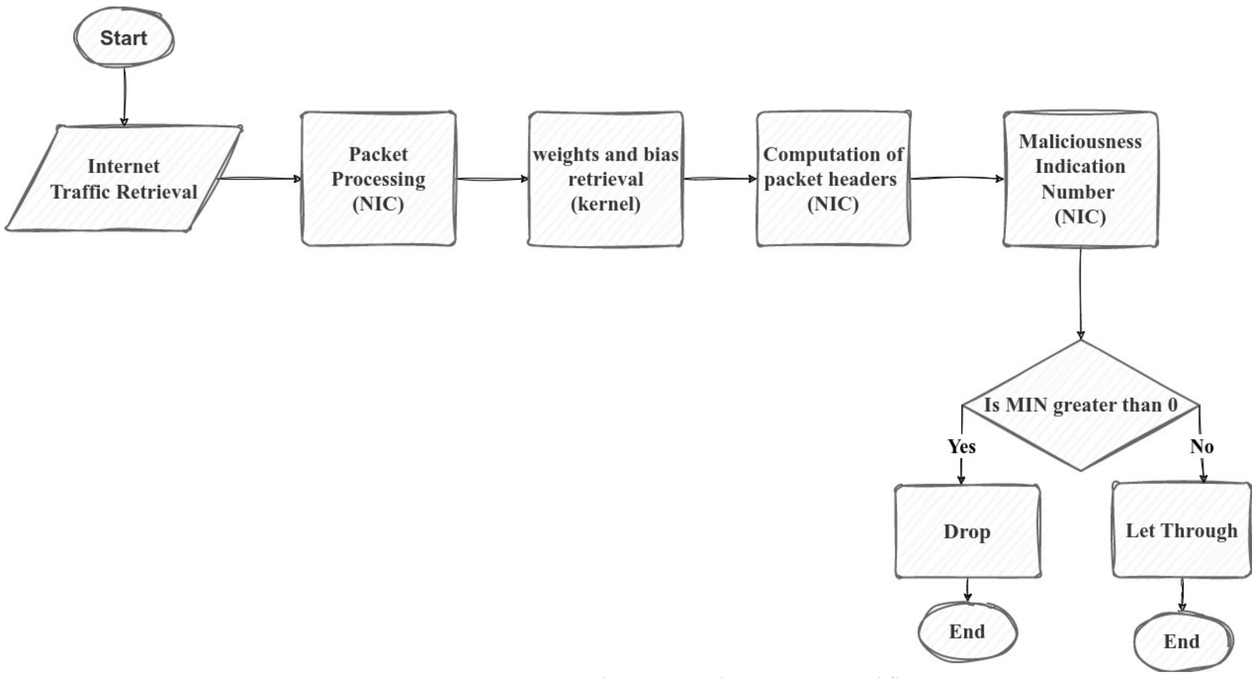

Start
Internet Traffic Retrieval
Packet Processing (NIC)
weights and bias retrieval (kernel)
Computation of packet headers (NIC)
Maliciousness Indication Number (NIC)
Is MIN greater than 0
Yes
No
Drop
Let Through
End
End

Figure 15 eIDPS intrusion detection and prevention workflow

4.8. Sequence Diagram

The diagram in Figure 13 depicts how the different components of the program work together. There are four columns, the user space, the kernel, the hardware, and the network traffic. The User-space column has two items: the ML model's weights and bias that get inserted into the eBPF program, and the hook script that is also inserted to the eBPF program in order to hook the program to the Kernel, which serves as the second column of the sequence diagram. It is shown that the kernel communicates with the third column, which is the Hardware, through the "**XDP_DROP"** and the "**XDP_PASS"** functions of the XDP program. These functions dictate if the packet should go into the system or not, based on the input of the weights that got inserted to the program from the ML model. The Hardware column is where the Network Interface Card is found. The NIC communicates with the last column, which is "Network Traffic", by allowing the incoming network packets in if the traffic is benign or simply dropping the packet assuming that it is malicious.

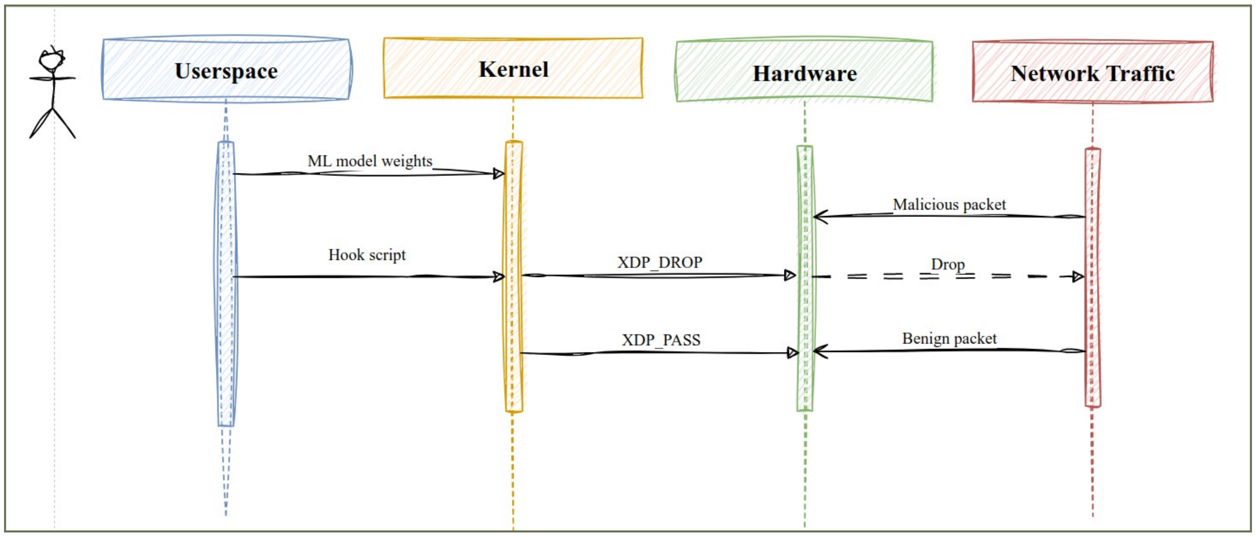


Figure 16 eIDPS Sequence Diagram

5. Evaluation

5.1. Experiments

In this section the eIDPS efficiency and the effectiveness in detecting and preventing malicious activities will be evaluated, namely, if the experiments that were conducted had the results that we expected.
The metrics were measured with **mpstat**[20] and **iostat**[21], which are pre-built programs in Kali distribution of Linux and generally in all Linux distributions. These tools measure the **CPU** and the **Input/Output metrics**, which are the operations performed by the system when it transports data, respectively. Furthermore, the packets that were transmitted and received were captured in both the attacker and the victim computer system's using Wireshark. Lastly, the training and validation accuracy and loss of the neural network model were measured during the training and validation processes, while they were also plotted into figures that are presented further below.

5.2. Methods - Scenarios

For the evaluation of the proposed solution two scenarios were developed, one with an already developed solution for network intrusion detection and prevention and one with the proposed eIDPS. The aim is to compare these scenarios to find the least heavy solution for system resources along with the most efficiency in preventing attacks. In addition to that, the ML model was trained before the experiments were conducted. Moreover, several cyber-attacks were selected for the conduction of the experiments, stemming from the following two attack categories: Network Reconnaissance and Distributed Denial of Service (DDOS).

5.2.1. Snort Scenario

In the first scenario the NIPS that was used was Snort with custom rules for dropping packets. For instance, the following rule detects a SYN stealth scan by comparing the incoming packet information, such as the flag of the packet and its signature id with values set in the rule;
"***drop tcp any any -> $HOME_NET any (msg:"Nmap SYN Scan detected"; flags:S; detection_filter:track by_src, count 5, seconds 10; sid:1000001; rev:1;)***".

If the information included in the packet are matched with the values set in the rule, then the packet is dropped. In addition to that, a custom plug-in was used that enabled Snort utilizing ML.
The same ML model and Dataset that was used with the eIDPS was also used with Snort. After validating all the rules, attacks were launched from a secondary Kali Linux operating system (VM). The following attacks were used: **SYN Flood**[22], **SYN scan**[23], **UDP Scan**[24], **XMAS Scan**[25]**,** and **OS Fingerprint**[26].

[20] https://man7.org/linux/man-pages/man1/mpstat.1.html
[21] https://man7.org/linux/man-pages/man1/iostat.1.html
[22] https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/
[23] https://nmap.org/book/synscan.html
[24] https://nmap.org/book/scan-methods-udp-scan.html
[25] https://nmap.org/book/scan-methods-null-fin-xmas-scan.html
[26] https://nmap.org/book/man-os-detection.html

5.2.2. eIDPS Scenario

In the second scenario, the proposed eIDPS solution was deployed for detecting and preventing malicious activities. Afterwards, the same attacks were launched from the secondary Kali system, namely **SYN Flood**, **SYN scan**, **UDP Scan**, **XMAS Scan,** and **OS Fingerprint**.

5.3. Conducting the Experiments

When the attacks were made in both scenarios, "***iostat 2***" and *"**mpstat -p ALL 1**" commands* were executed, to display the **I/O metrics** and the **CPU usage** on the terminal respectively. Furthermore, Wireshark was running on both the attacker and the victim side to display the transmitted and received packets.
Additionally, the training and validation accuracy and loss of the ML model were also measured during training and validation. While validation accuracy evaluates the model's effectiveness in generalizing on fresh, untested data, training accuracy analyzes how well the model performs on the training data. In other words, the training accuracy serves as a measure of how well the model fits the training data, while the validation accuracy assesses the model's ability to generalize and fit new, previously unseen data, both of which play vital roles in evaluating the model's performance and potential overfitting concerns.
Finally, the training and validation loss measurements displayed "**how efficiently**" the model predicts if a packet is malicious or benign, and "**how reliable**" its performance is on a separate validation dataset, that was not used in the training process.
Considering that Snort runs in the user space and detects and processes packets in this space, it is only logical that it will utilize more system resources than the proposed eIDPS solution, which runs directly in the kernel. More specifically, eBPF offloads the network traffic in the NIC, which drops the packets at a very early stage, after low level packet analysis. This enables the system to calculate very fast if a packet is malicious and drop it without using any of the system's resources.

5.4. Testbed

The testbed host computer had an Intel i7-3770 (with 4 cores - 8 threads) processor**,** 32 Gigabytes of RAM and a 500 Gigabyte Samsung 870 EVO solid state disk. In addition to that, the testbed included an ethernet Peripheral Component Interconnect Express (PCIe) network card, named NetLink BCM57781, that supported XDP and eBPF functionality, as some older NIC cards do not support these novel technologies.
The experiments were conducted between 2 VMs, utilizing the VirtualBox as hypervisor and the Kali Linux distribution as the operating system. Both VMs had assigned two virtual cores of processing power and 8 Gigabytes of RAM. Moreover, the VMs were connected with an internal network, to isolate them from outside network packets and possible interference.

5.5. Machine Learning training and validation results

As previously referred, the training and validation loss gives the user knowledge about the model's efficiency at predicting if the incoming packet is malicious or not. As it can be seen in Figure 14, after **100 epochs** the training and validation loss is at **0.02700%** and **0.02685%,** respectively, in both scenarios. This means that the loss of training and validation is minimal, consequently the model has great efficiency at predicting the network's traffic malicious intentions. Moreover, the validation line in the figure is under the training line, inferring that no overfitting or underfitting occurs.

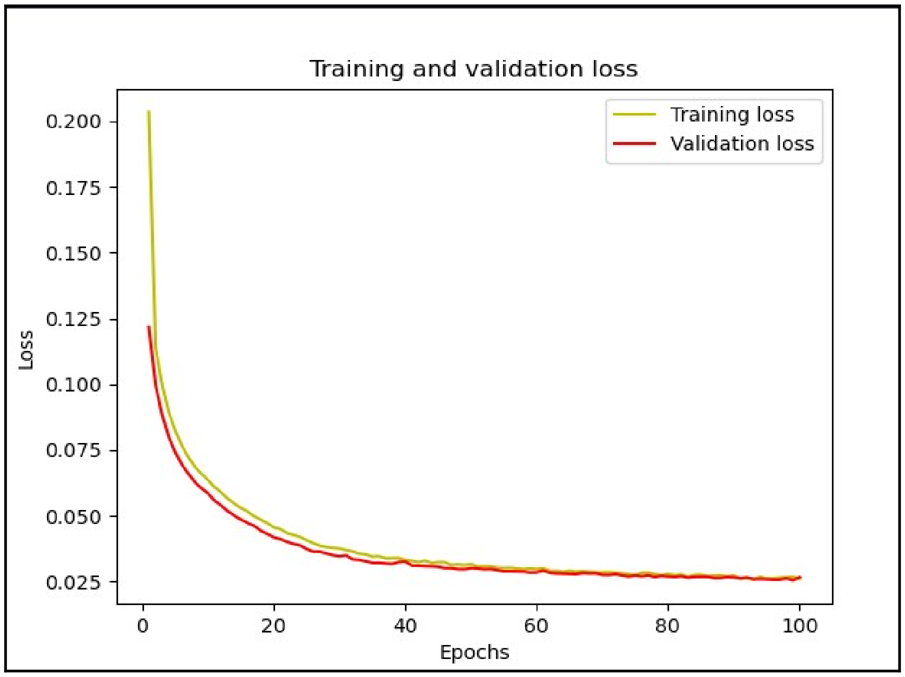


Figure 17 Training and Validation Loss

In Figure 15, the training and validation accuracy of the ML model are plotted. This figure provides information about the model's efficiency at learning the dataset and its performance when validating itself. The **training accuracy is 0.9912%** and the **validation accuracy is 0.98955%,** meaning that, in **100 epochs** ,in both scenarios, the model has a small amount of loss in training and validation. This indicates that the model is learning the dataset efficiently and performs exceptionally well in validating new data.

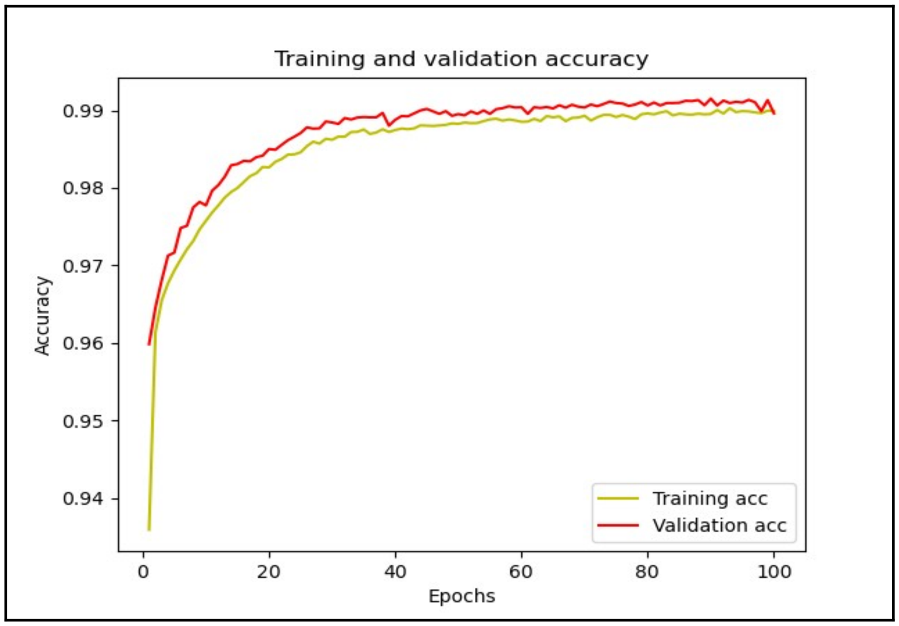


Figure 18 Training and validation accuracy

5.6. Experiment Results

In this section, a comprehensive evaluation of the experimental findings is presented to determine whether eIDPS is a better solution than a modified version of an open-source NIDS/NIPS.

5.6.1. Network Reconnaissance attacks

5.6.1.1. CPU Usage

In Figure 16, the attacks that were launched are depicted together with the CPU Usage on each attack regarding the Snort Scenario. In detail, the CPU percentage that Snort was using to detect and prevent the attacks was as follows:

1. **18.70%**, when it was attacked with an **“OS Fingerprint”** attack.
2. **15.60%**, when it was attacked with a **“SYN Scan”** attack.
3. **16.30%**, when it was attacked with an **“UDP Scan”** attack.
4. **23.80%**, when it was attacked with a **“XMAS Scan”** attack.

Regarding the eIDPS Scenario, the system’s resources cannot be accurately measured according to [4], because it is running in the kernel. One solution to this issue is to determine how many resources the whole kernel consumes when the attacks are happening. From these metrics one can estimate the true percentage of consumption. Consequently, the CPU usage of eIDPS is even less than the kernel’s as the eIDPS lies inside the kernel .
Regarding the same attacks as in the Snort Scenario, the CPU usage of the whole kernel is as follows:

1. **1,55%**, when it was attacked with an **“OS fingerprint**” attack.
2. **1,27%**, when it was attacked with a **“SYN Scan”** attack.
3. **1.51%**, when it was attacked with an **“UDP Scan”** attack.
4. **2.02%**, when it was attacked with a **“XMAS Scan”** attack.

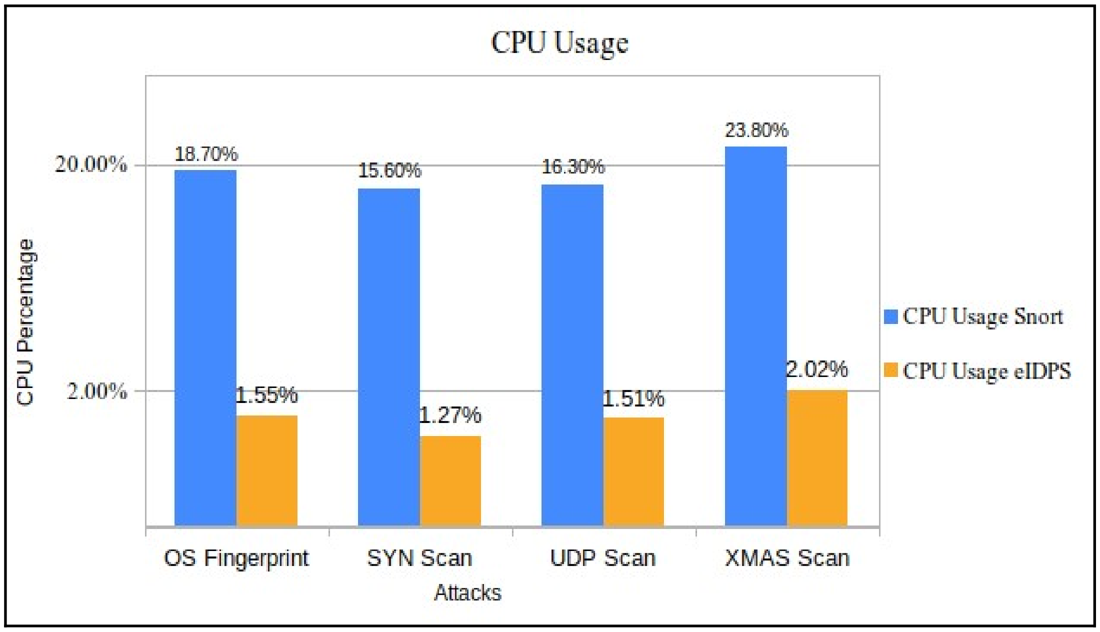


Figure 19 CPU Usage

The percentages of CPU usage observed in the Snort Scenario are vastly higher in comparison with the eIDPS Scenario, as it was anticipated. That is because the detection and prevention of the attacks in the Snort Scenario are happening simultaneously in the user space, thus more CPU is used to process the packet and then predict if it is malicious or not.

5.6.1.2. Packets passed into the system.

In Figure 18, the packets that were transmitted throughout different attacks were plotted together with the number of packets that passed into the system when Snort and eIDPS were deployed, respectively.
More specifically, eIDPS performed as follows:

1. **SYN scan** *attack*: it let through **290 packets out of 2002** that were transmitted.
2. **OS Fingerprint** *attack*: it let through **260 packets out of 2048** that were transmitted.
3. **UDP Scan** *attack:* it let through **644 packets out of 2032** that were transmitted.
4. **XMAS Scan** attack: it let through **270 packets out of 200** that were transmitted.

Snort on the other hand, did not perform as efficiently as eIDPS:

1. **SYN scan** *attack*: it let through **1103 packets out of 2002** that were transmitted.
2. **OS Fingerprint** *attack*: it let through **1003 packets out of 2048** that were transmitted.
3. **UDP Scan** *attack:* it let through **1678 packets out of 2032** that were transmitted.
4. **XMAS Scan** *attack*: it let through **1006 packets out of 2000** that were transmitted.

Finally, one interesting outcome is that Snort could hardly handle the **UDP Scan** as it allowed in the system **much more than 50%** of the packets that were transmitted.

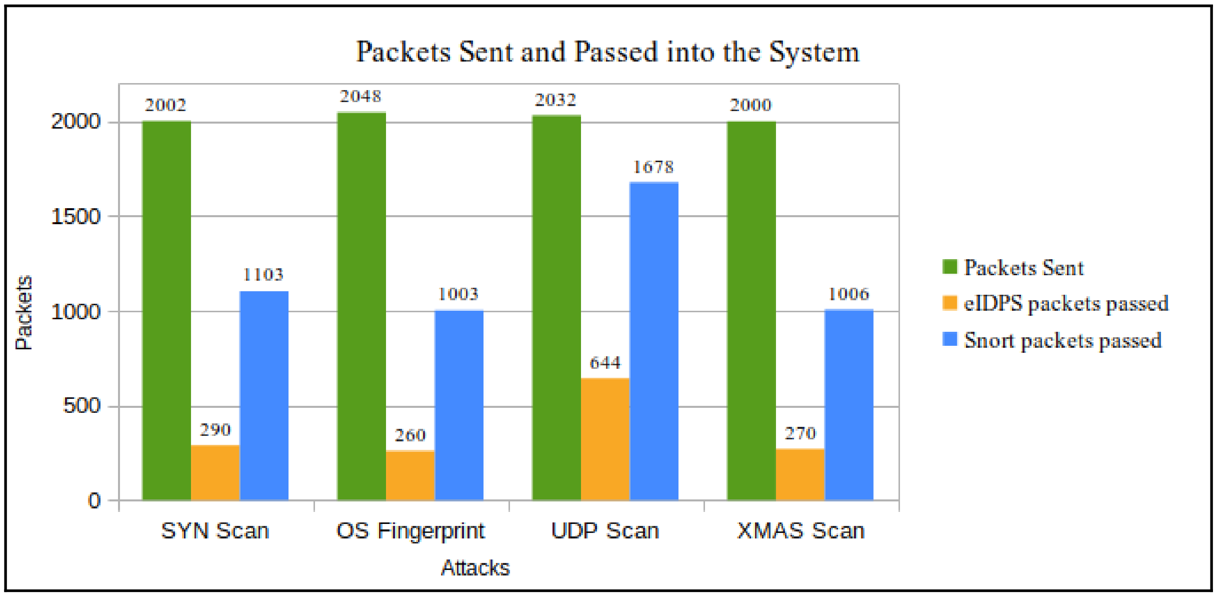


Figure 20 Packets Sent and Passed into the System

5.6.1.3. I/O Metrics

This section refers to the Input/Output (I/O) measurements taken for Snort and eIDPs. In the Table 1 below, someone can preview:

- the **transactions per second** (tps), which are the malicious requests that the packets were made,
- the **kilobytes per second** that was **read** (KB_read/s),
- the **kilobytes per second** that was **written** (KB_wrtn/s),
- the **kilobytes per second** that was **discarded** (KB_dscd/s) and
- the **total kilobytes per second** (from KB_read, KB_wrtn, KB_dscd).

Table 1 I/O Metrics

Attacks	tps		KB_read/s		KB_wrtn/s		KB_dscd/s		KB_read		KB_wrtn		KB_dscd	
	eIDPS	**Snort**	**eIDPS**	**Snort**	**eIDPS**	**Snort**	**eIDPS**	**Snort**	**eIDPS**	**Snort**	**eIDPS**	**Snort**	**eIDPS**	**Snort**
OS Fingerprint	2	3	0	0	12	34	0	0	0	0	24	68	0	0
SYN Scan	2.5	3	0	0	26	26	0	0	0	0	52	52	0	0
UDP Scan	2.5	3	0	0	24	64	0	0	0	0	48	128	0	0
XMAS Scan	2.5	2	0	0	26	270	0	0	0	0	52	540	0	0

In Table 1 above, it can be seen that, when the system got attacked while Snort and eIDPS were running, the **tps** always had a value as transactions were happening during the attack. Also, the system in both scenarios did not perform an input action in a storage device, so every value associated with **KB_read/s, KB_read, KB_dscd/s** and **KB_dscd** had a value of 0. However, as the table indicates, the system performed various output actions in the storage device; thus, resulting in the **KB_wrtn/s** and **KB_wrtn** columns to have values.

Particularly, when the system got:

1. "**OS Fingerprinted**", 3 transactions happened, 34 KB were written pers second and a total of 68 KB were written.
2. **"SYN Scanned"**, 3 transactions happened, 26 KB were written per second and 52 KB were written in total.
3. **"UDP Scanned"**, 3 transactions happened, 64 KB were written per second and 128 KB in total.
4. **"XMAS Scanned"**, 2 transactions happened, 270 KB were written per second and 540 KB in total.

In the same table it can be seen that the eIDPS program had less transactions per second and less KB were written in the storage device as it used buffers inside the kernel to store packets.

More specifically, when the system got:

1. **"OS Fingerprinted"**, 2 transactions happened, 12 KB were written per second and 24 KB in total.
2. **"Syn Scanned"**, 2.5 transactions happened, 26 KB were written per second and 52 KB in total.
3. "**UDP Scanned"**, 2.5 transactions happened, 24 KB were written per second and 48 KB in total.
4. **"XMAS Scanned"**, 2.5 transactions happened, 26 KB were written per second and 52 KB in total.

5.6.2. Distributed Denial of Service Experiment

5.6.2.1. Packets passed into the system

Distributed Denial of Service (DDOS) attack has to be analyzed separately, as there is not a way to prevent it entirely. We can only detect the attack and redirect the traffic somewhere else, as Cloudflare[27] had done by utilizing eBPF or using a third-party security module like Cloudflare[28], CISCO[29] etc., for DDOS attacks. For the scope of this thesis, we performed two DDOS attacks, one for the Snort Scenario and one for the eIDPS Scenario.
The DDOS attack, alternatively named ***SYN Flood*** attack, used the **hping3** tool for sending **100 packets** of **120 bytes per second**. In Figure 17, the packets that were sent and the packets that were passed into the system, in both scenarios, are depicted.
The result of this experiment depicted that Snort not only was unable to handle the ***SYN Flood*** attack and, therefore, it let all the packets through into the system, but also it could not even recognize that it was attacked. On the other hand, the proposed eIDPS solution was able to recognize that an attack was happening, and an endeavor was conducted to prevent the DDOS attack. In the end, eIDPS **dropped** around **34.000 packets**, in comparison with Snort that dropped **none.**
Last but not least, a CPU usage metric and an I/O metric would be useless, as Snort could not recognize that it was attacked. This means that it was idle throughout the attack.

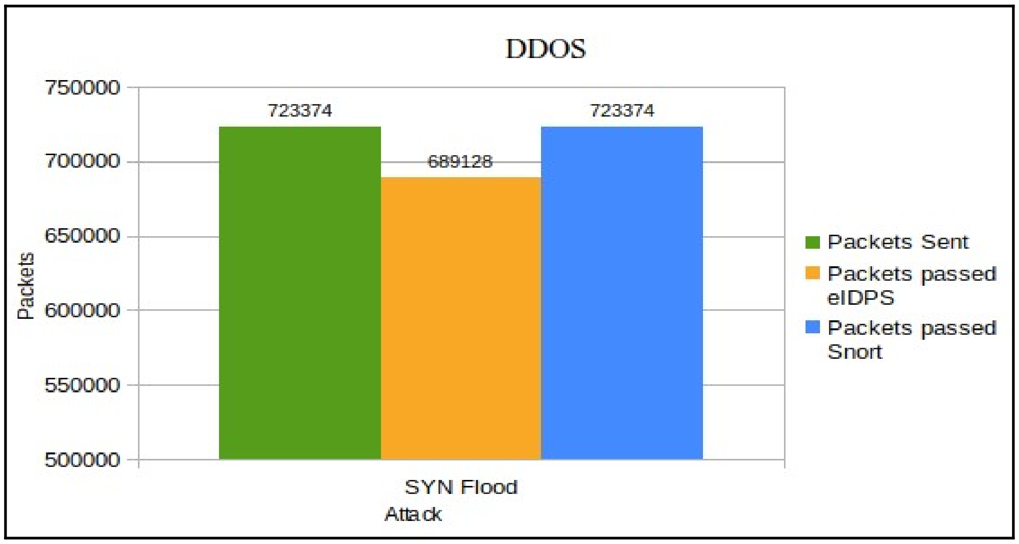


Figure 21 DDoS Attack

5.7. Evaluation Results Discussion

From the results of the experiments that were conducted it is showcased that the eIDPS uses less system resources and it is more efficient than Snort. As predicted, the lighter kernel-based program utilizes the fact that it is running inside the kernel and offloads the packet processing to NIC, so the packet processing happens a lot faster. It also uses less system resources as the kernel itself uses less resources opposed to applications that run in the user space. In addition to that, utilizing buffers inside the kernel for storing packets helped a lot in the I/O transactions that happened during the attack.

[27] https://blog.cloudflare.com/how-to-drop-10-million-packets/
[28] https://www.cloudflare.com/network-services/products/
[29] https://www.cisco.com/c/en/us/products/security/secure-ddos-protection/index.html

The ML aspect of the program has a pivotal role in identifying malicious packets with great efficiency. In the scenario where Snort was deployed, even though it was also utilizing the same ML model and dataset, it could not reach the efficiency that eIDPS had in packet processing, detection and in attacking prevention. Moreover, because it needed system resources to process the incoming packets and to decide whether to drop or not the packets after the processing in the user space, it resulted in a higher CPU usage, less efficiency in identifying the malicious packets and more I/O transactions.
Taking all the above into consideration, the results showed that the eIDPS is a better solution, in comparison with an open-source NIDS/NIPS, such as Snort, even if the latter utilized an ML algorithm for the detection of malicious traffic. eIDPS was observed as more lightweight and with greater efficiency in malicious packet prevention and detection.

6. Conclusion and Discussion

6.1. Aims and Summary

Nowadays, software applications that protect computer network systems are essential, as the danger of a cyber-attack is vastly high. eBPF is a novel technology that has already seen great growth and in the future, will be the go-to not only in cybersecurity, but also for load balancing, etc. The same can be said for ML as it is evolving day-by-day. With that in mind, this thesis proposed a system that is very lightweight and efficient in protecting a computer network system. After some Network Reconnaissance attacks (SYN Scan, OS Fingerprint, UDP Scan, XMAS Scan) and a DDOS attack (SYN Flood) were launched in two different security systems.
The eIDPS system had great results in detecting and dropping malicious packets, while it utilizes less system resources. On the other hand, the modified version of Snort was able to drop some malicious packets with some efficiency, as it also utilized ML, but in the bigger picture, not only used a lot more system resources than eIDPS, but also did not have the same efficiency in preventing attacks.

6.2. Limitations

Keeping in mind that is the proposed solution includes an eBPF-based program, there are some limitations, as mentioned earlier under the section "eBPF technology". In particular, there is a fixed number of maximum lines of code that can be used, at one million. Furthermore, it cannot have unbounded loops inside the source code and no memory relocation can happen. Regarding the dataset and the experiments, they were conducted in a closed environment with no outside interferences. Also, the used dataset was launched in 2017, so a more up to date dataset could be used.
In conclusion, there are ways to make the experiments more concrete. For example, a more accurate and up-to-date Dataset could be used for predicting malicious activities such as UGR'16[30] and CTU-13[31]. In addition, a wider range of attacks like malware, worms, DNS spoofing and man in the middle can be included and tested to solidify the results. Also, virtual machines with more CPU cores and threads can be used for the system resources metrics to be more accurate, as two or even four CPU cores are not enough for modern systems nowadays.
The eBPF technology and especially XDP have a lot of potential and will evolve more in the future. Direct packet processing inside the kernel is a great advantage in detecting and preventing packets and in utilizing less system resources over other solutions that run in the user space.

6.3. Future work

In the future, an enhancement of this thesis could be conducted, as several features and improvements could be added to both eBPF and in the Linux kernel. Consequently, there could be developed a more cultivated application that handles bigger network traffic and uses more intricated means for detection and prevention of malicious activities. Furthermore, in the ML aspect of the eIDPS new datasets can be used with the same ML model that has more modern cyber-attacks as entries.

[30] https://nesg.ugr.es/nesg-ugr16/
[31] https://www.impactcybertrust.org/dataset_view?idDataset=945

References

[1] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, “Survey of intrusion detection systems: techniques, datasets and challenges,” *Cybersecurity*, vol. 2, no. 1, p. 20, Dec. 2019, doi: 10.1186/s42400-019-0038-7.

[2] K. Radhakrishnan, R. R. Menon, and H. V Nath, “A survey of zero-day malware attacks and its detection methodology,” in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, IEEE, Oct. 2019, pp. 533–539. doi: 10.1109/TENCON.2019.8929620.

[3] M. Albahar, “Cyber Attacks and Terrorism: A Twenty-First Century Conundrum,” *Sci Eng Ethics*, vol. 25, no. 4, pp. 993–1006, Aug. 2019, doi: 10.1007/s11948-016-9864-0.

[4] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, “Network intrusion detection system: A systematic study of machine learning and deep learning approaches,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, Jan. 2021, doi: 10.1002/ett.4150.

[5] D. Stiawan, A. H. Abdullah, and Mohd. Yazid Idris, “The trends of Intrusion Prevention System network,” in *2010 2nd International Conference on Education Technology and Computer*, IEEE, Jun. 2010, pp. V4-217-V4-221. doi: 10.1109/ICETC.2010.5529697.

[6] S. Jose, D. Malathi, B. Reddy, and D. Jayaseeli, “A Survey on Anomaly Based Host Intrusion Detection System,” *J Phys Conf Ser*, vol. 1000, p. 012049, Apr. 2018, doi: 10.1088/1742-6596/1000/1/012049.

[7] A. Garg and P. Maheshwari, “A hybrid intrusion detection system: A review,” in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, IEEE, Jan. 2016, pp. 1–5. doi: 10.1109/ISCO.2016.7726909.

[8] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, “A taxonomy of computer worms,” in *Proceedings of the 2003 ACM workshop on Rapid malcode*, New York, NY, USA: ACM, Oct. 2003, pp. 11–18. doi: 10.1145/948187.948190.

[9] H. J. Highland, “A history of computer viruses — Introduction,” *Comput Secur*, vol. 16, no. 5, pp. 412–415, Jan. 1997, doi: 10.1016/S0167-4048(97)82245-6.

[10] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science (1979)*, vol. 349, no. 6245, pp. 255–260, Jul. 2015, doi: 10.1126/science.aaa8415.

[11] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, “A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection,” in *Proceedings of the 2003 SIAM International Conference on Data Mining*, Philadelphia, PA: Society for Industrial and Applied Mathematics, May 2003, pp. 25–36. doi: 10.1137/1.9781611972733.3.

[12] T. Condie, P. Mineiro, N. Polyzotis, and M. Weimer, “Machine learning for big data,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, June. 2013, pp. 939–942. doi: 10.1145/2463676.2465338.

[13] A. Garg and V. Mago, "Role of machine learning in medical research: A survey," *Comput Sci Rev*, vol. 40, p. 100370, May 2021, doi: 10.1016/j.cosrev.2021.100370.

[14] G. Apruzzese *et al.*, "The Role of Machine Learning in Cybersecurity," *Digital Threats: Research and Practice*, vol. 4, no. 1, pp. 1–38, Mar. 2023, doi: 10.1145/3545574.

[15] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal, "Creating Complex Network Services with eBPF: Experience and Lessons Learned," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, IEEE, Jun. 2018, pp. 1–8. doi: 10.1109/HPSR.2018.8850758.

[16] S. Jouet, R. Cziva, and D. P. Pezaros, "Arbitrary packet matching in OpenFlow," in *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, IEEE, Jul. 2015, pp. 1–6. doi: 10.1109/HPSR.2015.7483106.

[17] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System†," *Bell System Technical Journal*, vol. 57, no. 6, pp. 1905–1929, Jul. 1978, doi: 10.1002/j.1538-7305.1978.tb02136.x.

[18] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, New York, NY, USA: ACM, May 2010, pp. 495–504. doi: 10.1145/1806799.1806871.

[19] A. Koomsin and Y. Shinjo, "Running application specific kernel code by a just-in-time compiler," in *Proceedings of the 8th Workshop on Programming Languages and Operating Systems*, New York, NY, USA: ACM, Oct. 2015, pp. 15–20. doi: 10.1145/2818302.2818305.

[20] N. Van Tu, J.-H. Yoo, and J. W.-K. Hong, "Building Hybrid Virtual Network Functions with eXpress Data Path," in *2019 15th International Conference on Network and Service Management (CNSM)*, IEEE, Oct. 2019, pp. 1–9. doi: 10.23919/CNSM46954.2019.9012730.

[21] T. Høiland-Jørgensen et al., "The eXpress data path," in Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, Dec. 2018, pp. 54–66. doi: 10.1145/3281411.3281443.

[22] C. Xu, J. Shen, X. Du, and F. Zhang, "An Intrusion Detection System Using a Deep Neural Network with Gated Recurrent Units," IEEE Access, vol. 6, pp. 48697–48707, 2018, doi: 10.1109/ACCESS.2018.2867564.

[23] S. Waskle, L. Parashar, and U. Singh, "Intrusion Detection System Using PCA with Random Forest Approach," in 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), Jul. 2020, pp. 803–808. doi: 10.1109/ICESC48915.2020.9155656.

[24] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep Learning Approach for Intelligent Intrusion Detection System," IEEE Access, vol. 7, pp. 41525–41550, 2019, doi: 10.1109/ACCESS.2019.2895334.

[25] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. C. Júnior, and L. F. M. Vieira, "Fast Packet Processing with eBPF and XDP," ACM Comput Surv, vol. 53, no. 1, pp. 1–36, Jan. 2021, doi: 10.1145/3371038.

[26] M. Bachl, J. Fabini, and T. Zseby, “A flow-based IDS using Machine Learning in eBPF,” ACM Comput Surv, vol. 53, no. 1, pp. 1–36, Jan. 2021

[27] S.-Y. Wang and J.-C. Chang, “Design and implementation of an intrusion detection system by using Extended BPF in the Linux kernel,” Journal of Network and Computer Applications, vol. 198, p. 103283, Feb. 2022, doi: 10.1016/j.jnca.2021.103283.

[28] L. Caviglione, W. Mazurczyk, M. Repetto, A. Schaffhauser, and M. Zuppelli, “Kernel-level tracing for detecting stegomalware and covert channels in Linux environments,” Computer Networks, vol. 191, p. 108010, May 2021, doi: 10.1016/j.comnet.2021.108010.

[29] A. Dhamdhere, M. Luckie, B. Huffaker, kc claffy, A. Elmokashfi, and E. Aben, “Measuring the deployment of IPv6,” in *Proceedings of the 2012 Internet Measurement Conference*, New York, NY, USA: ACM, Nov. 2012, pp. 537–550. doi: 10.1145/2398776.2398832.

[30] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, “Performance Implications of Packet Filtering with Linux eBPF,” in 2018 30th International Teletraffic Congress (ITC 30), Sep. 2018, pp. 209–217. doi: 10.1109/ITC30.2018.00039.

[31] S. Miano, R. Doriguzzi-Corin, F. Risso, D. Siracusa, and R. Sommese, “Introducing SmartNICs in Server-Based Data Plane Processing: The DDoS Mitigation Use Case,” IEEE Access, vol. 7, pp. 107161–107170, 2019, doi: 10.1109/ACCESS.2019.2933491.

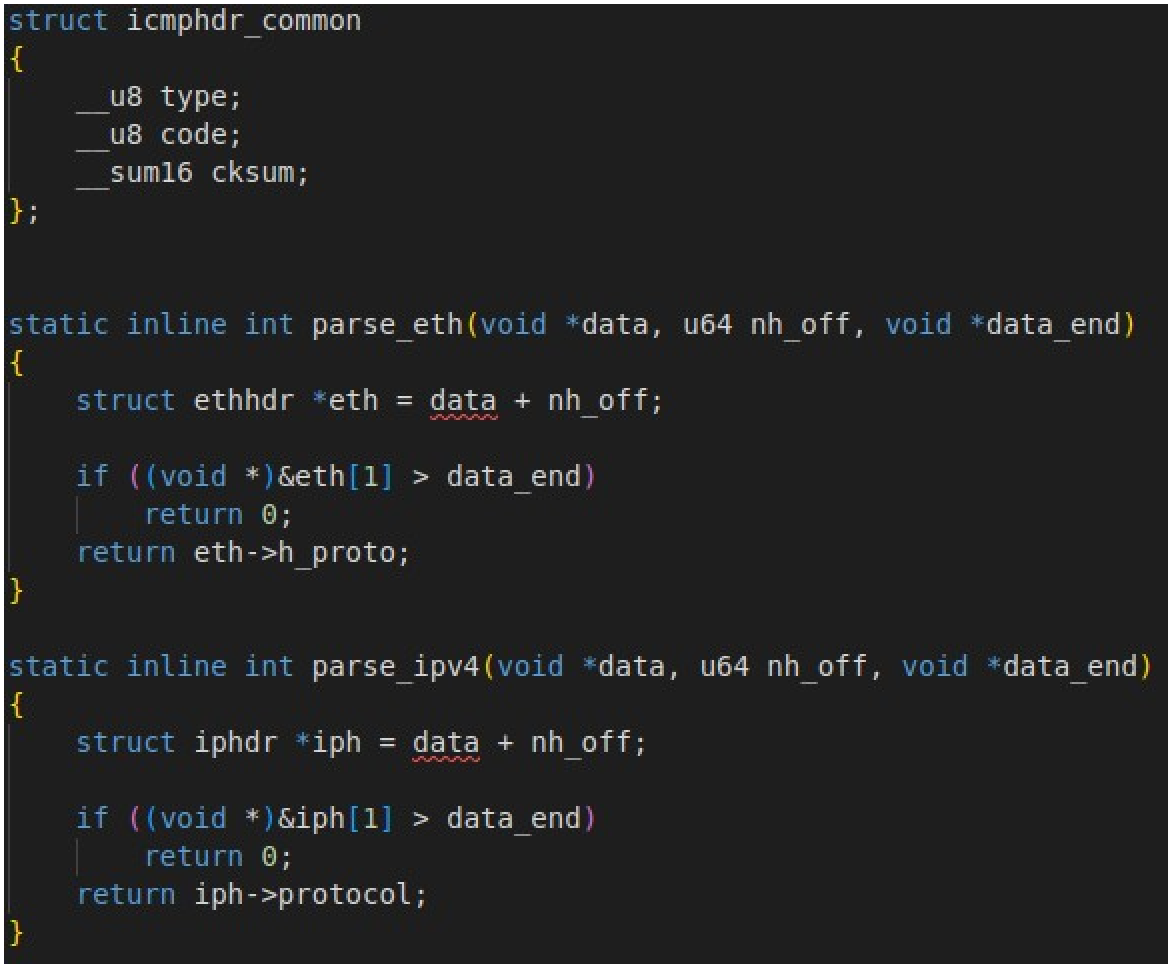
[32] O. Hohlfeld, J. Krude, J. H. Reelfs, J. Ruth, and K. Wehrle, “Demystifying the Performance of XDP BPF,” in 2019 IEEE Conference on Network Softwarization (NetSoft), Jun. 2019, pp. 208–212. doi: 10.1109/NETSOFT.2019.8806651.

[33] N. Rusk, “Deep learning,” *Nat Methods*, vol. 13, no. 1, pp. 35–35, Jan. 2016, doi: 10.1038/nmeth.3707.

[33] R. Yuste, “From the neuron doctrine to neural networks,” *Nat Rev Neurosci*, vol. 16, no. 8, pp. 487–497, Aug. 2015, doi: 10.1038/nrn3962.

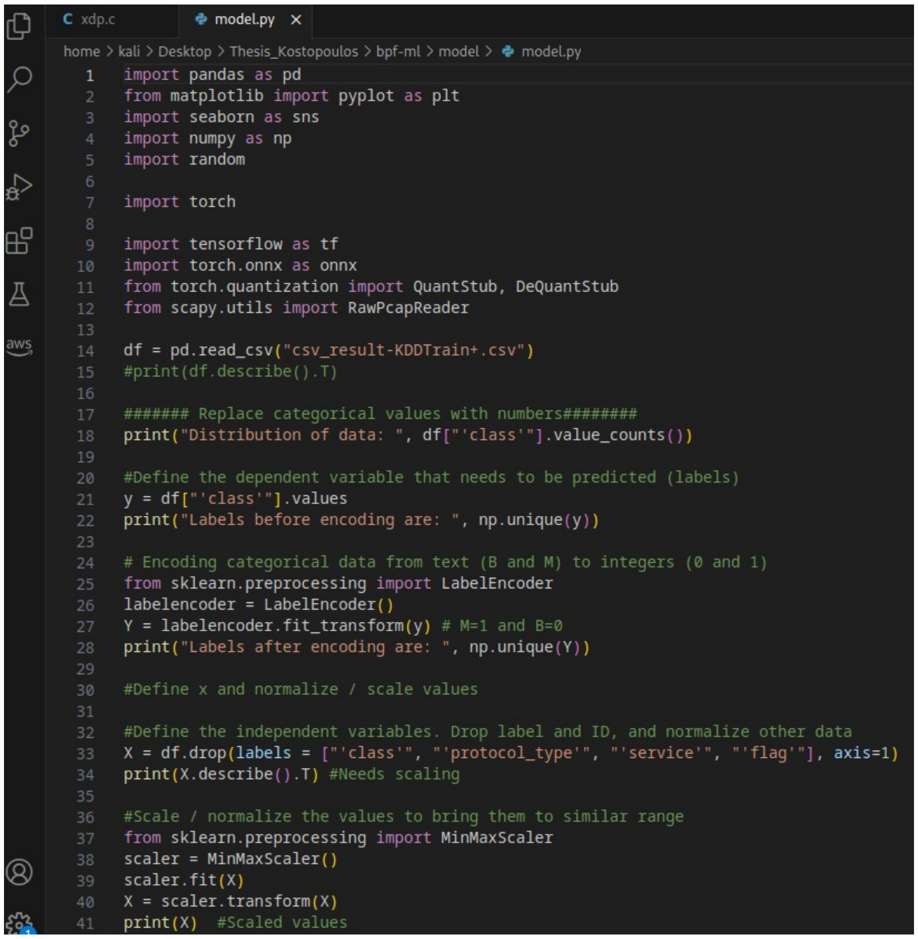
[34] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, IEEE, Jul. 2009, pp. 1–6. doi: 10.1109/CISDA.2009.5356528.

Appendix I - eBPF Code



```
struct icmphdr_common
{
    __u8 type;
    __u8 code;
    __sum16 cksum;
};


static inline int parse_eth(void *data, u64 nh_off, void *data_end)
{
    struct ethhdr *eth = data + nh_off;

    if ((void *)&eth[1] > data_end)
        return 0;
    return eth->h_proto;
}

static inline int parse_ipv4(void *data, u64 nh_off, void *data_end)
{
    struct iphdr *iph = data + nh_off;

    if ((void *)&iph[1] > data_end)
        return 0;
    return iph->protocol;
}
```

Appendix II - User Space Code


xdp.c
model.py
home > kali > Desktop > Thesis_Kostopoulos > bpf-ml > model > model.py

```python
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
import random

import torch

import tensorflow as tf
import torch.onnx as onnx
from torch.quantization import QuantStub, DeQuantStub
from scapy.utils import RawPcapReader

df = pd.read_csv("csv_result-KDDTrain+.csv")
#print(df.describe().T)

####### Replace categorical values with numbers########
print("Distribution of data: ", df["'class'"].value_counts())

#Define the dependent variable that needs to be predicted (labels)
y = df["'class'"].values
print("Labels before encoding are: ", np.unique(y))

# Encoding categorical data from text (B and M) to integers (0 and 1)
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
Y = labelencoder.fit_transform(y) # M=1 and B=0
print("Labels after encoding are: ", np.unique(Y))

#Define x and normalize / scale values

#Define the independent variables. Drop label and ID, and normalize other data
X = df.drop(labels = ["'class'", "'protocol_type'", "'service'", "'flag'"], axis=1)
print(X.describe().T) #Needs scaling

#Scale / normalize the values to bring them to similar range
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X = scaler.transform(X)
print(X)  #Scaled values
```