International Journal of Engineering Research in Current Trends (IJERCT)
ISSN: 2582-5488, Volume-3 Issue-5, October 2021

Machine Leaning for Flow Based Intrusion Detection Using Extended Berkley Packet Filter

Pratap Singh Pradhan
School of Computer Science and Technology, LNCT University, Bhopal, pspspsingh@gmail.com

Dr Praveen Kumar Mannepalli
School of Computer Science and Technology, LNCT University, Bhopal, praveen.hawassa@gmail.com

Abstract— **The extended Berkley packet filter (eBPF) is a novel technology that allows portions of code to be dynamically loaded into the Linux kernel. Because it allows the kernel to process some packets without the intervention of a userspace programme, it can considerably speed up networking. eBPF has so far been utilised for simple packet filtering applications like firewalls and DDoS defence. We show that a flow-based network intrusion detection system based on machine learning may be developed entirely in eBPF. Our method employs a decision tree to determine if a packet is malicious or not, taking into account the complete preceding context of the network flow. When compared to the same solution implemented as a userspace programme, a speed boost of over 19% has been achieved.**

Keywords— eBPF, Firewall, Decision tree, DDoS, Packet

I. INTRODUCTION

A. Extended Berkeley Packet Filter

The Extended Berkeley Packet Filter (eBPF) is a Linux kernel instruction set and execution environment. It allows for runtime change, interactivity, and kernel programmability. The eXpress Data Path (XDP), a kernel network layer that processes packets closer to the NIC for faster packet processing, can be programmed using eBPF. Developers can develop programmes in C or P4 and then compile them to eBPF instructions, which the kernel or programmable devices can handle (e.g., SmartNICs). eBPF has been quickly embraced by big firms such as Facebook, Cloudflare, and Netronome since its launch in 2014. Network monitoring, network traffic manipulation, load balancing, and system profiling are examples of use cases. eBPF may be dynamically injected during runtime and is well tested to ensure that it does not crash or get stuck in infinite loops. However, this verification is only viable for non-turing complete programmes. As a result, eBPF programmes cannot have loops of any length; instead, loops must always have a maximum number of iterations. Backward leaps in the code are also often prohibited. As a result, eBPF can only be used to implement algorithms that aren't turing-complete. The majority of eBPF programmes are written in C and then compiled to eBPF bytecode. This eBPF bytecode is dynamically compiled to native code after being injected into the kernel.

eBPF is particularly well suited to packet processing: Certain actions, such as discarding a packet, can be performed when it arrives at a network interface. This is useful for applications that record packets according to specified criteria, such as firewalls or tcpdump. If only packets from port 80 are to be recorded, for example, tcpdump will construct an eBPF programme that encodes this and loads it into the kernel [1]. All packets that do not meet the filter will be dropped by the kernel, and only the correct ones will be sent to tcpdump. Alternatively, tcpdump may receive all packets and filter them on its own. The disadvantage is that each packet must then be handed from the kernel to tcpdump, which requires transferring the entire packet into memory as well as additional computing steps. As a result, whenever possible, passing packets between the kernel and programmes should be avoided for performance reasons. This is possible using eBPF only.

Because the eBPF bytecode is compiled to native code, it should run as quickly as any other kernel code. However, because eBPF is verified, it can only use a limited number of data structures. Normal C arrays, for example, cannot be used in an eBPF programme because they allow outof-bounds accesses. For example, even though an array of length 10 only has 10 elements, C would give access to the 15th element. As a result, eBPF programmes make advantage of secure data structures. However, because checking the boundaries of an array each time it is accessed necessitates more CPU effort, this could result in a performance cost.

Kernel modules are an alternative to using eBPF. Kernel modules, on the other hand, have the disadvantage of not being able to be tested for reliability and having to be compiled for a certain kernel version. Furthermore, designing kernel modules is not easy, and it is frequently impossible to enhance particular kernel capabilities with a kernel module without modifying the kernel itself. Changing the kernel as a whole necessitates recompiling the kernel , which is inconvenient.

B. Machine Learning for IDS:

Artificial neural networks (ANN), support vector machines (SVM), K-nearest neighbour (KNN), nave Bayes, logistic regression (LR), decision trees, clustering, and mixed and hybrid approaches are among the conventional machine learning models (shallow models) for IDS. Some of these solutions have been studied for decades and have a well-developed methodology. They are concerned not only with the detection effect, but also with practical issues such as detection efficiency and data administration. As per previous work in this field following given algorithm of machine learning and respective improvement in algorithms has been raised in Table 1.

Table 1: Machine learning Algorithms for IDS

Algorithms	**Advantages**	**Improvement Measures**
ANN	Can work Non-Linear Data, more fitting ability	Optimizers, Activation Function, Loss Function

KNN	Quick Training, Noise robustness, Massive data handling	PSO (Particle Swarm Optimization), SMOTE
SVM	Generation Capability, Can work well on small Dataset	PSO
Naïve Bayes	Noise robustness, incremental learning	Can import latent variable
Decision Tree	Auto feature selection, strong interpretation	Balanced Dataset , Introducing latent variables
K - Means	Rapid training, scalability, big data can fit.	Initialization method

Some researchers have already looked into eBPF-based IDSs or considered applying machine learning in eBPF. Propose that eBPF be used to discover performance problems using AI. They do, however, merely present a concept and do not evaluate the benefit of their technique, and they also employ eBPF to build solutions for preventing Denial-of-Service assaults. They don't employ machine learning.

The fact that eBPF is not turing complete is one of the drawbacks of utilising it for machine learning. ML techniques, such as decision trees (DTs) and neural networks (NNs), on the other hand, do not require loops in their implementation and hence do not require turing-completeness, and thus can be implemented in eBPF. Because tree-based approaches are a simple and effective ML method for IDSs , we chose to employ DTs. As discussed in the preceding sections, eBPF data structures require more processing than standard C data structures [8]. As a result, we'd like to answer the following question in this study: Is eBPF faster than a solution implemented as a regular userspace programme? The disadvantage of a userspace programme is that all packets must be transmitted between the kernel and the programme, which is slow. The downside of the eBPF programme is that it uses potentially slower data structures [3]. As a result, it'll be fascinating to see if eBPF can be faster in reality, even for complicated algorithms that regularly use data structures.

II. METHOD AND EVALUATION PATH

We imagine a method that maintains track of each network flow and analyses each packet in relation to the flow's prior packets. Certain assaults, for example, may not be noticed until the fourth packet of the network flow containing the attack arrives. We employ eBPF's built-in hash tables for this purpose, which allow us to store information for each flow. The conventional five tuple of protocol type, source and destination IP, and source and destination port is used as the key. A general idea of proposed method of cleaning program has been given in figure 1 using eBPF user program and kernel.

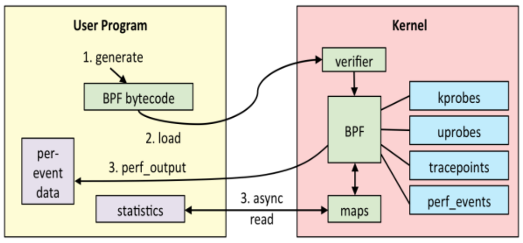

Figure 1: eBPF filter user and kernel

Because these features have proven useful for network traffic analysis, we use the source and destination ports, the protocol identifier (UDP, TCP, ICMP, etc.), the packet length, the time since the last packet of the flow, and the packet direction (from sender to receiver or vice versa) as features. We also give the average packet size, the duration since the last packet, and the direction for all packets received thus far in the flow. Because the standard deviation cannot be estimated in eBPF due to the lack of sophisticated arithmetic operations such as the square root operation, the mean absolute deviation is computed for these three features as well. The fact that eBPF only supports integers rather than floating point operations is a concern. The eBPF program execution is safe and using filter, optimal working using less data set is more efficient under learning environment.

The popular CIC-IDS-2017 dataset is used. We use scikit-learn to train the decision tree with a maximum depth of 10 and a maximum number of leaves of 1100, with a train/test split of 70:30 ratio, and it achieves a 97.78 percent accuracy on the testing dataset after training. This precision is equivalent to that attained in related research [4].

We make all of the source code and other materials from this work openly available to facilitate replication and encourage future exploration by other researchers. We employ the previously trained DT and implement the same IDS in both userspace and eBPF. The code is identical except for the data structures, which are different since, as previously stated, many standard data structures are not useable in eBPF. Furthermore, several data structures in eBPF, such as hash maps, are not available by default in a standard C userspace programme [1], thus the userspace version uses a rudimentary hash map implementation from the Linux kernel.

We use Linux network name spaces to simulate a network.For this, we'll use a switch to connect a server and a client, and we'll set the links to have no delay in addition to the delay generated by the Linux kernel when forwarding packets. In addition, we set the maximum link speed to infinity. iPerf is used to connect the server and the client. iPerf will run as quickly as feasible because the network speed is unrestricted. The greatest speed is simply limited by the computer's ability to process packets quickly. The IDS is installed by opening a raw socket on the server's network interface. As a result, all packets passing through the server are routed through the raw. The IDS decodes the socket and processes it. The IDS can be implemented as a traditional userspace application or through the use of eBPF, and they all operate in parallel (not at the same time). Both implementations are run for a total of 10 seconds. Instead of installing the IDS on a client computer, it can be installed on a router or switch that runs a recent Linux version and can thus run eBPF.

Table 2 Packets per implementation (Max over 10 run each)

packets/s	Userspace		eBPF	
	Mean	**SD**	**Mean**	**SD**
	125 320	2527	152 174	1191

The userspace implementation inspects less packets per second (125320) than the eBPF implementation, as shown in Table 2. (152174). As a result, the eBPF implementation is more than 19% faster than the userspace counterpart.

III. DISCUSSION

An interesting thing to ponder is that for complicated ML models, the expense of employing eBPF's specific data structures eventually outweighs the value that eBPF provides. Future research could, for example, look into whether random forests (RFs) or deep neural networks (NNs) can likewise obtain a performance advantage when used in eBPF. However, we were able to demonstrate that for a simple decision tree model, eBPF delivers a considerable performance benefit in this study. The work can be extended with muti aspect streams where each record in constant time and constant memory is required. Again main focus may be expected on feature selection on dataset where optimal retrieval techniques using cascading techniques can be used.

REFERENCES

[1] SF. Risso and M. Tumolo, “Towards a faster Iptables in eBPF,” 2018..

[2] I. Ben-Yair, P. Rogovoy, and N. Zaidenberg, “AI & eBPF based performance anomaly detection system,” in SYSTOR, ACM, May 2019.

[3] H. M. Demoulin, I. Pedisich, N. Vasilakis, V. Liu, B. T. Loo, and L. T. X. Phan, “Detecting Asymmetric Application-layer Denialof-Service Attacks In-Flight with FineLame,” USENIX, 2019.

[4] H. van Wieren, “Signature-Based DDoS Attack Mitigation: Automated Generating Rules for Extended Berkeley Packet Filter and Express Data Path,” 2019.

[5] Y. Choe, J.-S. Shin, S. Lee, and J. Kim, “eBPF/XDP Based Network Traffic Visualization and DoS Mitigation for Intelligent Service Protection,” in Advances in Internet, Data and Web Technologies, Springer, 2020.

[6] F. Iglesias, D. C. Ferreira, G. Vormayr, M. Bachl, and T. Zseby, “NTARC: A Data Model for the Systematic Review of Network Traffic Analysis Research,” Applied Sciences, Jan. 2020.

[7] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” in ICISSP, SCITEPRESS, 2018.

[8] A. Hartl, M. Bachl, J. Fabini, and T. Zseby, “Explainability and Adversarial Robustness for RNNs,” in BigDataService 2020, IEEE, Apr.2020.