An APT Attack Detection Method Based on eBPF and Transformer

Rixuan Qiu[1,2], Hao Luo[1], Sitong Jing[3], Xinxiu Li[1], and Yuancheng Li[1]

(Corresponding author: Yuancheng Li)

School of Control and Computer Engineering, North China Electric Power University[1]

No. 2 Beinong Road, Changping District, Beijing, China

Information & Telecommunication Branch of State Grid Jiangxi Electric Power Supply Co., Ltd.[2]

No. 7077 Changdong Avenue, High-tech Zone, Nanchang, Jiangxi Province

PowerChina Jiangxi Electric Power Engineering Co., Ltd.[3]

No. 426, Jingdong Avenue, Qingshanhu District, Nanchang City, Jiangxi Province

Email: ncepua@163.com

(Received Oct. 21, 2023; Revised and Accepted May 20, 2024; First Online July 30, 2024)

Abstract

Advanced persistent threats (APTs) are a type of attack that uses advanced techniques to launch long-term and targeted network attacks against specific entities. APTs can exploit system vulnerabilities and use sophisticated and stealthy methods to evade detection by traditional means. This paper proposes an APT attack detection system based on eBPF and Transformer to address this challenge. The system leverages eBPF to efficiently collect network traffic feature data from the bottom layer of the Linux kernel network stack and then applies a Transformer-based deep learning model to identify APT attacks. The paper conducts experiments in a simulated network environment and compares the performance of the system with existing attack detection methods. The results show that the system achieves a recognition accuracy of 96.5%, has high operational efficiency, and can effectively detect APT attacks in network systems, providing a new solution to cope with such attacks.

Keywords: APT Attack Detection; eBPF; Transformer

1 Introduction

Advanced Persistent Threats (APTs) are typically orchestrated by large enterprises, sovereign states, and affiliated entities with the primary motive of political or economic gains, specifically targeting a particular organization or institution. These attacks are characterized by their long-term objectives, as they aim to infiltrate the entire system rather than focusing on short-term interests. APTs exhibit traits such as extended activity cycles, extensive concealment, and significant destructive potential. In many cases, the victims of APT attacks remain unaware of the intrusion until after the damage has occurred. Common targets of APT attacks include multinational corporations, international organizations, and government departments. Notably, critical infrastructures such as electric power and energy facilities, which play a vital role in national defense and public welfare, are among the primary targets susceptible to APT attacks [2, 4, 6, 7, 10, 12, 13, 16, 18, 21].

The kernel's packet filter, known as the CMU/Stanford Packet Filter (CSPF), was initially proposed by Mogul and others [19]. This system offers a packet filtering mechanism within user space. Subsequently, the Berkeley Packet Filter (BPF) was introduced by Steven McCanne and Van Jacobson at the Usenix conference in 1993 [15]. BPF's main concept involves capturing network packets at the link layer within the operating system's network stack, filtering the packets, and delivering the filtered data to the relevant application for processing. Unlike CSPF, which relies on a memory stack mechanism, BPF utilizes registers and employs a non-shared cache model for each process, resulting in improved performance during packet filtering. BPF's introduction led to the development of numerous related applications, such as the widely used network debugging and traffic analysis tool tcpdump, and the pivotal network data analysis library function libpcap, both of which employ BPF instructions. BPF has become an integral component of underlying development.

However, BPF has limitations that hinder its further expansion in application scenarios. For instance, due to its design based on Reduced Instruction Set Computing (RISC), BPF registers are generally 32-bit and cannot fully utilize the capabilities of modern 64-bit CPU registers. In 2014, Alexei Starovoitov made improvements to the BPF instruction set, introducing the extended Berkeley Packet Filter (eBPF) [1, 3, 5, 8, 9, 11, 14, 17, 20]. One significant change was the support for 64-bit register operations, expanding the number of registers from 2 to 10.

Another important feature introduced was just-in-time (JIT) compilation, allowing binary code injection into the kernel at runtime. These enhancements not only broadened the usage scenarios for eBPF but also significantly improved performance. eBPF was first introduced in version 3.18 of the Linux Kernel and has found widespread applications in networking, kernel debugging, and optimization.

Given eBPF's excellent filtering capabilities, its study in network applications holds significant importance. The core idea in eBPF network applications involves forwarding filtered data packets to other applications for processing, thereby reducing the system overhead caused by a large influx of packets into the network protocol stack. By employing appropriate algorithms and solutions, eBPF can further advance the prevention of DDoS attacks and intrusion prevention systems, contributing to enhanced network security.

This research paper leverages the distinctive attributes of eBPF and employs a Transformer-based approach to present and advance an APT attack detection method specifically designed for Linux networks. Firstly, the method captures comprehensive network traffic information within the Linux kernel network stack, encompassing crucial details such as network request types, IP datagram headers, TCP segment headers, and UDP user datagram headers. Subsequently, a classification model is constructed using a Transformer neural network to conduct an in-depth analysis of the accumulated data, facilitating the identification of potential APT attacks. The primary objectives of this study are outlined as follows:

Initially, leveraging eBPF and its associated technologies, a tool is developed to capture network traffic data at the lowest level of the Linux kernel network stack. This tool enables the meticulous and cost-efficient extraction of network traffic data from the Linux kernel, subsequently transferring it to the relevant program interface in user mode for utilization.

Furthermore, a highly efficient detection module is constructed to process the extracted kernel data. This module utilizes a deep learning detection model based on the Transformer architecture, which, when integrated with the network traffic data extraction tool powered by eBPF, establishes a novel system for detecting abnormal network traffic.

In summary, this method can be broadly categorized into two primary components: the data acquisition module and the analysis and detection module. The data acquisition module is responsible for gathering and preprocessing data during the operation of the Linux system kernel, while the analysis and detection module is employed for data training, analysis, and APT attack detection. Further details are provided below:

2 System Design and Implementation

APT Attack Detection Using eBPF and Transformer

This chapter presents the system design and implementation. The system consists of two components: data collection and processing, and data analysis. The system uses eBPF to collect network traffic data and the Transformer to analyze the data and detect APT attacks. The system logic and data flow are illustrated in Figure 1.

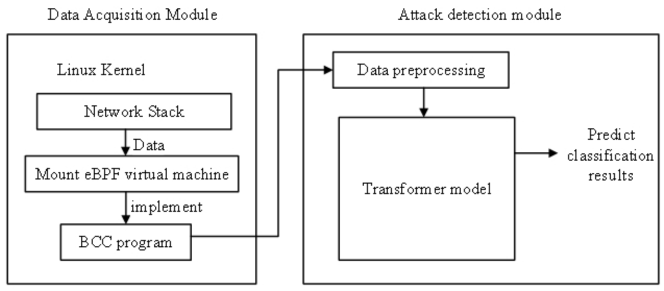

Figure 1: Logic and data flow

The system operates in two modules. The first module is the data acquisition module, which runs on the Linux kernel and uses the eBPF virtual machine and related programs. This module collects network traffic data and sleeps for 10 seconds between each collection. The second module is the attack detection module, which receives the data from the first module, processes it, feeds it into a classification model, and predicts and classifies the data based on the model. The system then outputs the results and returns to the data acquisition module. The system alternates between these two modules periodically. The following will provide a detailed introduction to these two modules.

3 Data Acquisition Module

3.1 eBPF

The history of eBPF (extended Berkeley Packet Filter) technology can be traced back to its predecessor, BPF (Berkeley Packet Filter). Initially, BPF was developed as a network packet filtering technology implemented in Unix systems through the kernel's virtual machine. As time passed, BPF was expanded to include system tracing and performance analysis capabilities, leading to its implementation in the Linux kernel. Throughout this evolution, BPF gradually transformed into a highly flexible and powerful virtual machine technology, culminating in the development of eBPF.

eBPF, an extension of BPF, specifically targets the Linux kernel, enabling dynamic modification of kernel behavior at runtime by executing custom BPF programs within the kernel. Brendan Gregg proposed eBPF in 2014 intending to broaden the scope of BPF to encompass various applications, including high-performance packet filtering, dynamic tracing, container security, and service monitoring. Compared to traditional BPF, eBPF introduces numerous new features and optimizations, supporting different program types such as user space and kernel space programs. Moreover, eBPF allows safe loading and unloading of programs at runtime.

At the core of eBPF technology lies the compilation of specific code into binary bytecode, referred to as an eBPF program. These programs can then be executed in the kernel and intercepted by hook functions, enabling monitoring, filtering, processing, and forwarding of system calls, network data packets, and events. The key advantage of eBPF is that it can operate in the kernel space safely and efficiently without necessitating modifications to the kernel source code or recompilation. This flexibility empowers eBPF to monitor and control kernel behaviors such as data filtering, security auditing, and performance analysis in a scalable manner.

The flexibility and scalability of eBPF programs are made possible by just-in-time compilation technology, which converts BPF bytecode into native machine code. This approach allows eBPF programs to dynamically generate and optimize code at runtime, adapting to varying kernel behaviors and environments.

While eBPF technology allows for the collection of kernel operating data through the placement of observation points within Linux kernel functions, its practical application can be complex. It involves a series of actions such as code writing, kernel loading, result data retrieval, unloading, and exiting, which can make data retrieval from the kernel space cumbersome. Consequently, a new generation of eBPF-based tools has emerged in various domains, including networking, security, application configuration/tracing, and performance troubleshooting. These tools operate independently of existing kernel functionality while maintaining execution efficiency and security, actively modifying runtime behavior in specific scenarios. Among these tools, BCC holds a fundamental and pivotal role.

3.2 BCC

BCC (BPF Compiler Collection) is an open-source toolkit that offers a comprehensive range of robust tools and libraries for effectively working with eBPF (extended Berkeley Packet Filter) programs. Its primary objective is to streamline the development, debugging, and analysis of eBPF-based applications across various domains, including networking, performance monitoring, and kernel tracing. The BCC toolkit consists of a collection of high-level frontends and libraries that enable developers to write eBPF programs using familiar programming languages such as C, Python, and Lua. It provides an intuitive interface that simplifies the compilation and loading of eBPF programs into the kernel, enabling users to fully harness the capabilities of eBPF for tasks such as network analysis, system monitoring, and troubleshooting.

One prominent aspect of BCC is its extensive repertoire of pre-built tools and utilities. These tools cater to a wide array of use cases, encompassing packet filtering, network tracing, performance profiling, and kernel function tracing. Leveraging eBPF programs, BCC tools facilitate the capture and analysis of network packets, monitoring of system events, and diagnosis of performance issues, all without the need for intrusive instrumentation or kernel modifications.

In essence, BCC catalyzes simplifying eBPF development, empowering users to leverage the potential of eBPF for efficient and flexible system monitoring, analysis, and troubleshooting endeavors.

3.3 Sysdig

Sysdig is a comprehensive system monitoring, analysis, and troubleshooting tool that offers open-source universal system visibility with native container support. While the Linux platform already provides several tools for system performance analysis, such as strace, tcpdump, htop, iftop, lsof, and netstat, as well as various logging and monitoring tools, Sysdig stands out due to its exceptional integration, power, and flexibility.

3.4 Module Design

The data collection module implemented in this method utilizes Sysdig for data acquisition. Initially, Sysdig is deployed within the Linux kernel container, after which the sysdig command is executed via the command line interface to observe the system's present status and activities. By employing various filters and output formats, traffic data is obtained. Additionally, the BCC toolset is employed to invoke the compiler, thus converting Python code into eBPF bytecode. Through the aid of relevant helper functions, the eBPF storage data structure map is accessed, allowing for mapping to the corresponding Python data structure. Consequently, the data acquired through the eBPF approach can be fed into the deep learning module developed in Python. The specific structure is illustrated in the accompanying Figure 2.

The process begins by leveraging XDP technology to attach the eBPF program to the Linux kernel network stack. Once the network card collects the data packet and places it into the kernel buffer, the XDP program retrieves the pointer and length of the data packet from the kernel buffer. This information is encapsulated as a "ctx" pair, providing access to and manipulation of the content and metadata of the data packet through the "ctx" object. Additionally, the program utilizes pointers to indicate the starting and ending addresses of the region where the Ethernet frame, passed from the device driver layer, is stored. By employing the structure that represents the Ethernet frame's header, the program sequentially unpacks the network traffic, traversing from the Ethernet frame to the IP data packet, and subsequently to the TCP /UDP data

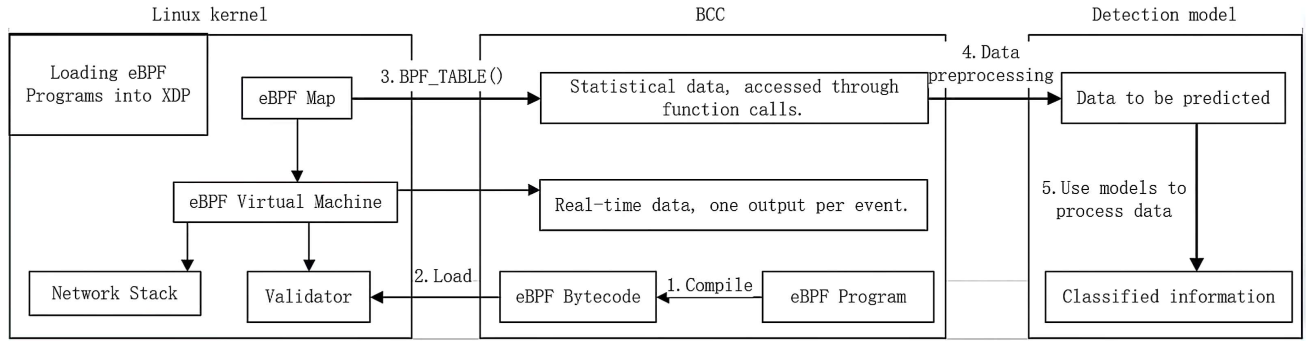

Figure 2: Schematic diagram of system operation logic

Table 1: Feature details

Feature Value	Type	Description
prot_type	Computational	The most frequent protocol of packets in a unit of time
src_bytes	Count	The average number of bytes from the source host to the destination host
dst_bytes	Count	The average number of bytes from the destination host to the source host
flag_count	Computational	The number of packets with the SYN, ACK, PSH, URG, and RST flags set to 1
src_ip_count	Count	The number of distinct source IP addresses.
pkt_length	Count	The average length of packets.
pkt_count	Count	The number of packets in a unit of time (used for calculating the rate).
tcp_pktt_cnt	Count	The number of TCP packets
tcp_sport_cnt	Count	The number of source ports in TCP packets
tcp_dport_cnt	Count	The number of destination ports in TCP packets
tcp_fflag_cnt	Count	The number of TCP packets with the FIN flag set to 1

packet, allowing for the extraction of the data layer by layer.

Subsequently, standard network attack analysis techniques are employed to derive features from the captured network traffic. This approach follows the methodology proposed by Karimazad, and the extracted feature details are presented in Table 1.

4 Attack Detection Module

4.1 Transformer

The present article incorporates elements of the Transformer model, necessitating a brief introduction. The Transformer model, renowned for its ability to enhance training efficiency through the employment of attention mechanisms, stands as a pioneer in leveraging this approach in deep learning models. It comprises two distinct modules, namely the encoder and decoder, enabling effective handling of lengthy textual challenges. The Transformer model's exceptional performance has extended its applications beyond natural language processing, finding widespread utilization in the realm of computer vision as well.

4.2 Attention Mechanism

The attention mechanism serves as a pivotal component within the Transformer model, facilitating the extraction of vector relationships through the computation of weight matrices. Its specific implementation involves the conversion of input word vectors into a feature vector denoted as Z.

$$Q = X \times W^Q$$

$$K = X \times W^K$$

$$V = X \times W^V$$

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

In this context, the input vector sequence is denoted as X, while W^Q, W^K, W^V represents a randomly initialized matrix, and d_k signifies the vector dimension. The weight matrices Q, K, and V correspond to the Query, Key, and Value, respectively. By considering the embedding vector as a query vector alongside multiple key-value pairs, distinct weights are attained through individual computations and subsequently applied to the corresponding value. This mechanism, known as self-attention, derives the weight matrix directly from the embedding vector itself. The attention mechanism conducts diverse weight calculations based on contextual information, assigning higher weights to significant words and lower weights to

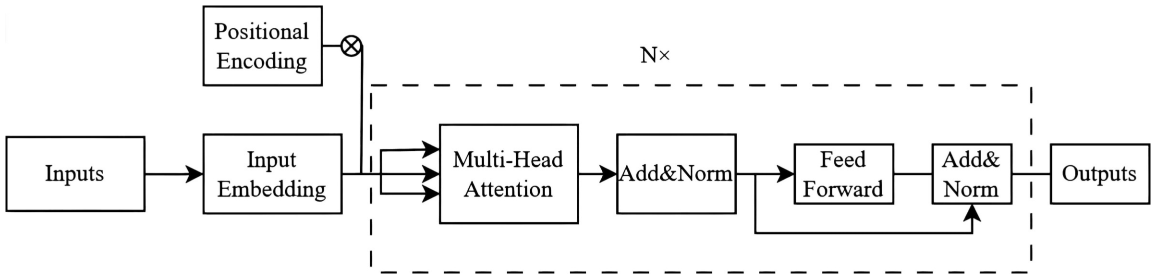

Figure 3: encoder blocks of Transformer

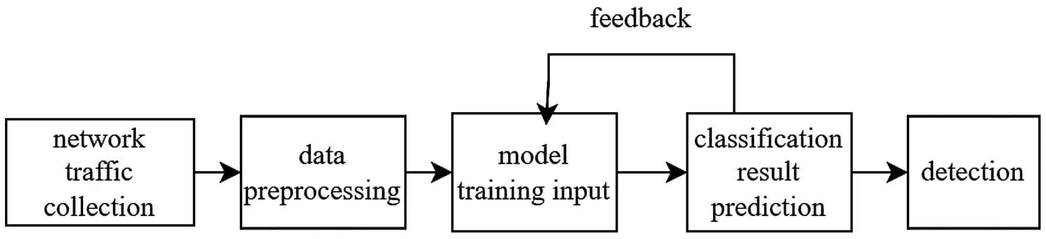

Figure 4: APT attack detection process

less significant ones, thereby encapsulating the mutual relationships between words within the obtained weight vector. Consequently, the attention mechanism effectively captures the attributes of lengthy text sequences while preserving their semantic information.

4.3 Model Structure

This approach utilizes multiple encoder blocks from the Transformer model for data training. The distinguishing characteristic of the Transformer model lies in its utilization of the attention mechanism. This mechanism incorporates the Q, K, and V matrices to compute weights, which are then applied to the sum operation to generate the output of self-attention. This process effectively extracts data features. The specific structure is illustrated in Figure 3.

4.4 Process of Attack Detection

Utilizing the aforementioned model, the designed process for APT attack detection, as depicted in Figure 4, is as follows.

Initially, internal data containing APT attacks, gathered from the Linux kernel, is annotated based on their distinguishing characteristics. Subsequently, the data undergoes preprocessing to transform it into feature vector format, which is then utilized for training the model. Through the training process, the model's parameters are updated based on the prediction outcomes, aiming to enhance accuracy and achieve effective APT attack detection.

5 Experiment

To assess the efficacy of the technique, we will conduct an experiment aimed at replicating a real-world network environment. The experiment entails generating various types and intensities of attack traffic while comparing different detection solutions. Key metrics such as performance, real-time capabilities, and accuracy will be employed to evaluate the prototype system. To ensure the smooth execution of the experiments, it is essential to have the requisite hardware resources and establish a suitable experimental environment. We will utilize a total of five servers, out of which two will function as web application servers. These servers will be configured with identical specifications, including Intel(R) Core(TM) i7-12700F processors, CentOS 7 64-bit operating system, and appropriate web application and system monitoring tools. Both the prototype system from this study and other comparable detection solutions will be individually installed. The remaining three servers will be designated as DDoS attack servers and will simulate HTTP DDoS attacks. These servers will run on Windows Server 2019 and will be equipped with the PyLoris attack tool. The five servers will operate on a separate local area network (LAN).

5.1 Experiment Specific Settings

To evaluate the performance of the Transformer-based detection prototype system proposed in this paper, we conducted a series of experiments with the following settings. We used five servers, a LAN environment, a PyLoris attack tool, and a Snort system. We deployed the prototype system and the Snort system on two Web application servers, respectively, and verified that they were configured correctly and running normally. We deployed the PyLoris tool on three DDoS attack servers and set different attack parameters to simulate various types and intensities of HTTP DDoS attacks. We also ran a simulation program on a client host to mimic the normal user access to the Web application server.

We launched HTTP DDoS attacks in stages according to the experimental design. In each stage, we used one DDoS attack server to initiate a specified type and intensity of attack, and observed how the prototype system and the Snort system detected and defended against it. Meanwhile, we started the client host that simulated normal user access under different attack scenarios and recorded whether the prototype system and the Snort system misclassified them as attacks or not. We also recorded the success rate of normal user access.

5.2 Comprehensive Evaluation Experiment Based on Snort

Snort is a network intrusion detection and prevention system (NIDS/NIPS) that monitors and analyzes network traffic in real-time, and detects and alerts on malicious packets based on predefined rules. Snort can operate in three modes: as a packet sniffer, a packet logger, or a full-fledged NIDS/NIPS. Snort is open source and free to use for both individuals and organizations. Snort's rules are categorized into two types: community rules and subscription rules. Community rules are developed by the Snort community and quality tested by the Cisco Talos team. They are freely available to all users. Subscription rules are developed, tested, and approved by the Cisco Talos team. They are only accessible to paid subscribers in real-time. Snort also supports Open App ID, a feature that enables application identification and control. Therefore, we choose Snort for comparative experiments.

To configure Snort for HTTP DDoS detection and prevention, we need to use a set of rules that can identify and block potential network attacks. We use some common HTTP DDoS rules that are based on the characteristics of attack traffic, such as request rate, request type, source IP address, and so on. Table 2 shows some examples of these rules. For instance, we can create rules to detect excessive request rates within a certain time window, or to detect abnormal User-Agent identifiers. After defining the rules, we apply them to Snort and start the system to monitor network traffic. When Snort matches traffic with our rules, it generates alerts and takes predefined defensive actions. These actions may include blocking attack source IP addresses and limiting request rates. By monitoring and responding in real-time, Snort can effectively protect the system and network resources from HTTP DDoS attacks.

We followed the same experimental settings and procedures as the comprehensive evaluation experiment of the prototype system proposed in this paper but used Snort as a comparable HTTP DDoS detection and prevention solution. We configured Snort with a set of rules for HTTP DDoS attacks and monitored network traffic in real-time. We observed how Snort detected and defended against HTTP DDoS attacks of different types and intensities. After the experiment, we analyzed Snort's performance in terms of detection accuracy, resource consumption, detection delay, and attack resistance, and evaluated its applicability in real-world scenarios.

5.3 Comprehensive Evaluation Experiment

In this section, we conduct a comprehensive evaluation experiment on the APT attack detection prototype system based on eBPF and Transformer algorithms, which is proposed in this article. We adjust the classification conditions according to the experimental results to obtain the best performance of the prototype system in detecting and defending against different types or intensities of DDOS attacks and normal user access.

Before the comprehensive evaluation, we need to build a complete prototype system and train the attack detection model. We divide the data set into two parts: one is the normal user network traffic data set, which is labeled to remove the non-normal user data; the other is the HTTP DDOS attack traffic data set, which is filtered to retain only the HTTP DDOS data. We merge the two parts of the data and train the model after data cleaning and feature extraction processes. We test the trained classifier model and load it into the kernel through the classifier loader. We fine-tune the three classification conditions during the testing process to achieve optimal results, including lower server resource usage and higher classification accuracy.

We start the experiment after ensuring that all hardware resources and the experimental environment are ready, including five servers, LAN environment, attack tools, and Snort system. We deploy the prototype system and Snort system on two web application servers respectively and verify that they can run normally with correct configuration parameters. We deploy the PyLoris tool on three APT attack servers and set the attack parameters to simulate different types and strengths of attacks during the experiment. We run a pre-written simulation program on the client host to simulate the behavior of a normal user accessing the web application server. We launch an HTTP DDOS attack in stages according to the experimental design. In each stage, we use a DDOS attack server to launch an attack of a specified type and intensity, and then observe the detection and defense performance of the prototype system and Snort system. We also start a client host that simulates a normal user and observes whether the prototype system and Snort system misclassify the normal user access behavior under different attack scenarios. We record the success rate and response time of normal user access. Finally, we use perf-tools to collect data on various evaluation indicators in real-time, including detection accuracy, resource consumption, detection

Table 2: Part of the snort rule table

Rule Description	Content
Detecting Slow Http Connection Attacks	alert tcp any any->any 80(msg:"Possible HTTP DDoS Slowloris attack detected";flow:to_server;content: "GET";nocase;content:"HTTP/1.1";nocase; detection_filter:track by_src,count 50,seconds 120; classtype:attempted-dos; sid:1000001;rev:1;)
Detect Abnormal Http Request Methods	alert tcp any any->any 80(msg:"Possible HTTP DDoS Abnormal HTTP Request Method";flow:established; content:"—0d 0a—";within:10; pcre: "/^[\x20-\x26\x28-\x7e]+\x20/";classtype: attempted-dos; sid:1000002;rev:1;)
Detect A Large Number Of Identical User-Agent Requests In A Short Period Of Time	alert tcp any any->any 80(msg:"Possible HTTP DDoS High rate of identical User-Agent requests" ; flow:established; content: "User-Agent—3a—" ; nocase;threshold: type threshold, track by_src,count 50, seconds 5; classtype:attempted-dos; sid:1000003;rev:1;)

delay, and anti-DDoS attack capabilities, to ensure the accuracy and reliability of data collection.

After the experiment is completed, we obtain the measurement indicators through the following calculation methods:

1) Detection accuracy: We define True Positive (TP) as correctly identified attacks, True Negative (TN) as correctly identified normal traffic, False Positive (FP) as false positive normal traffic, and False Negative (FN) as false negative attacks. We calculate the detection accuracy (Accuracy) as follows: Accuracy = (TP + TN) / (TP + TN + FP + FN).

2) Resource consumption: We monitor the CPU usage and memory usage of the prototype system regularly during the experiment. We calculate the average CPU usage and memory usage as follows: Average CPU usage = Sum of CPU usage / Number of measurements; Average memory usage = Sum of memory usage / Number of measurements. We can obtain this data using performance monitoring tools provided by the operating system or third-party monitoring tools.

3) Detection delay: We record the time (T1) when each attack starts and the time (T2) when the system successfully detects the attack and takes action. We calculate the detection delay as follows: Delay = T2 - T1. We average all the delays to obtain the average detection delay.

4) Defense capabilities: We record the number of successful attempts and the total number of attempts by normal users to access the web application server during the attack. We calculate the access success rate as follows: Success rate = Number of successes / Total number of attempts. We also record the response time of normal users accessing the web application server. We calculate the average response time as follows: Average response time = Sum of response times / Number of successes.

5.4 Analysis of The Experimental Results

We conducted the same comprehensive evaluation experiment on the prototype system of this article and Snort, and collected and analyzed the experimental data. In this section, we compare the performance of the two systems in detail based on various evaluation indicators. Figure 6 shows the detection accuracy of different types of HTTP DDOS attacks by the prototype system and Snort. The experimental results show that both systems have high accuracy in detecting different types of HTTP DDOS attacks. However, the prototype system outperforms Snort in three types of attacks: GET/POST type slow message body attacks, asymmetric requests, and large payload POST requests. Especially for asymmetric requests, the prototype system achieves an accuracy of 96.30%, which is significantly higher than Snort's 82.80%. On the other hand, for HTTP flood-type attacks, the prototype system has a slightly lower accuracy of 97.90%, compared to Snort's 98.00%. Overall, the prototype system demonstrates high accuracy in detecting most types of attacks.

Figure 5 shows the comparison of detection accuracy between the prototype system and Snort under different types of APT attacks. The prototype system has higher detection accuracy than Snort in general, which can be attributed to the Transformer algorithm used by the prototype system. Unlike Snort's rule-based detection method, which relies on pre-defined rule sets and may fail to recognize certain types of attacks, the Transformer algorithm can effectively extract attack information and adapt to emerging attack characteristics through learning and analysis of historical data. Therefore, the Transformer algorithm can make more accurate judgments on attacks and reduce false positives or false negatives. In

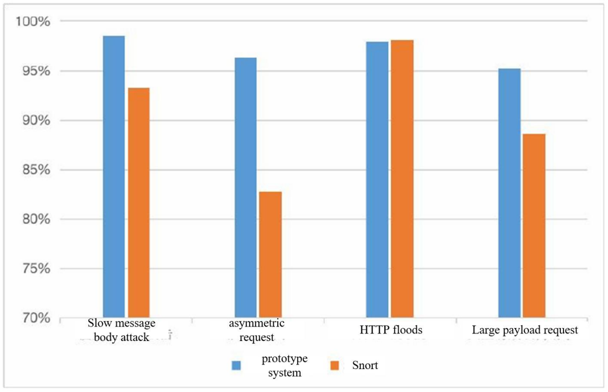

Figure 5: Different Types Of Attack Detection Accuracy

summary, the experiments in this article demonstrate that the HTTP DDOS detection method proposed in this article is more efficient than the rule-based Snort, as evidenced by the higher detection accuracy of the prototype system in various HTTP DDOS attack scenarios.

Another measurement indicator is the detection latency of different types of HTTP DDOS attacks. The lower the detection latency, the faster the alarm and response processing speed can be achieved. Figure 5 illustrates the detection latency of different types of HTTP DDOS attacks by the prototype system and Snort. According to the experimental results, the prototype system has lower detection latency than Snort in dealing with different types of HTTP DDOS attacks.

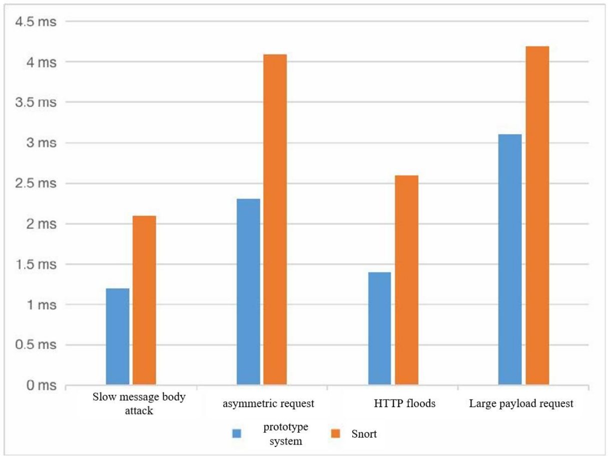

Figure 6: The detection latency of different types of HTTP DDOS attacks

6 Conclusions

This paper presents an attack detection method based on eBPF and Transformer for advanced persistent threats (APTs) in the network and designs a prototype system to conduct a series of experimental evaluations. The paper compares the performance of the prototype system with the rule-based Snort system in terms of detection accuracy, detection latency, defense capability, and user experience. The experimental results show that the prototype system outperforms Snort in all aspects, achieving significantly higher detection accuracy, lower detection latency, higher normal user access success rate, and lower response time. The paper concludes that the proposed method can effectively detect and defend against APTs and is superior to existing methods, providing a feasible solution in the field of network security.

Acknowledgments

This work was supported in part by the State Grid Jiangxi Information & Telecommunication Company Project "Research on key technologies for edge access security in the new power system intelligent IoT system" under Grant SGJXXT00DDJS2310143. The authors gratefully acknowledge the anonymous reviewers for their valuable comments.

References

[1] Tahir Alyas, Sikandar Ali, and Habib Ullah Khan, "Container performance and vulnerability management for container security using docker engine," 2022.

[2] Sidahmed Benabderrahmane, Ghita Berrada, James Cheney, and Petko Valtchev, "A rule mining-based advanced persistent threats detection system," *CoRR*, vol. abs/2105.10053, 2021.

[3] Kelly Brady, Seung Moon, Tuan Nguyen, and Joel Coffman, "Docker container security in cloud computing," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0975–0980. IEEE, 2020.

[4] Jun Dong, Dongran Liu, Xihao Dou, Bo Li, Shiyao Lv, Yuzheng Jiang, and Tongtao Ma, "Key issues and technical applications in the study of power markets as the system adapts to the new power system in china," *Sustainability*, vol. 13, no. 23, p. 13409, 2021.

[5] Ana Duarte and Nuno Antunes, "An empirical study of docker vulnerabilities and of static code analysis applicability," in *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*, pp. 27–36. IEEE, 2018.

[6] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," *arXiv preprint arXiv:2001.01525*, 2020.

[7] Katharina Hofer-Schmitz, Ulrike Kleb, and Branka Stojanović, "The influences of feature sets on the de-

tection of advanced persistent threats," *Electronics*, vol. 10, no. 6, p. 704, 2021.

[8] Omar Javed and Salman Toor, "An evaluation of container security vulnerability detection tools," in *Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing*, pp. 95–101, 2021.

[9] Omar Javed and Salman Toor, "Understanding the quality of container security vulnerability detection tools," *arXiv preprint arXiv:2101.03844*, 2021.

[10] George Karantzas and Constantinos Patsakis, "An empirical assessment of endpoint detection and response systems against advanced persistent threats attack vectors," *Journal of Cybersecurity and Privacy*, vol. 1, no. 3, pp. 387–421, 2021.

[11] Fang Li, *Network Security Evaluation and Optimal Active Defense based on Attack and Defense Game Model.* 2023.

[12] Jinxin Liu, Yu Shen, Murat Simsek, Burak Kantarci, Hussein T Mouftah, Mehran Bagheri, and Petar Djukic, "A new realistic benchmark for advanced persistent threats in network traffic," *IEEE Networking Letters*, vol. 4, no. 3, pp. 162–166, 2022.

[13] Xiangde Luo, Minhao Hu, Tao Song, Guotai Wang, and Shaoting Zhang, "Semi-supervised medical image segmentation via cross teaching between cnn and transformer," *arXiv e-prints*, 2021.

[14] Akalanka Mailewa, Susan Mengel, Lisa Gittner, and Hafiz Khan, "Mechanisms and techniques to enhance the security of big data analytic framework with mongodb and linux containers," *Array*, vol. 15, p. 100236, 2022.

[15] Antony Martin, Simone Raponi, Théo Combe, and Roberto Di Pietro, "Docker ecosystem–vulnerability analysis," *Computer Communications*, vol. 122, pp. 30–43, 2018.

[16] Zijie Meng, Xinlei Cai, Jinzhou Zhu, and Xu Lin, "Study on the influence of extreme weather on power grid operation under new power system," in *Proceedings of the 2021 4th International Conference on Algorithms, Computing and Artificial Intelligence*, pp. 1–6, 2021.

[17] Anupama Mishra, Neena Gupta, and Brij. B. B. Gupta, "Defensive mechanism against ddos attack based on feature selection and multi-classifier algorithms," *Telecommunication systems: Modeling, Analysis, Design and Management*, 2023.

[18] Y. Pan, T. Zhou, J. Zhu, and Z. Zeng, "Construction of apt attack semantic rules based on att & ck," *Journal of Cyber Security*, vol. 6, no. 3, pp. 77–90, 2021.

[19] Elochukwu Ukwandu, Mohamed Amine Ben-Farah, Hanan Hindy, Miroslav Bures, Robert Atkinson, Christos Tachtatzis, Ivan Andonovic, and Xavier Bellekens, "Cyber-security challenges in aviation industry: A review of current and future trends," *Information*, vol. 13, no. 3, 2022.

[20] Marcos A. M. Vieira, Matheus S. Castanho, Racyus D. G. Pacífico, Elerson R. S. Santos, and Luiz F. M. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys*, vol. 53, no. 1, pp. 1–36, 2020.

[21] Satya Prakash Yadav, Subiya Zaidi, Annu Mishra, and Vibhash Yadav, "Survey on machine learning in speech emotion recognition and vision systems using a recurrent neural network (rnn)," *Archives of computational methods in engineering: State of the art reviews*, no. 3, p. 29, 2022.

Biography

Ri-xuan Qiu works for the Information communication Branch of State Grid Jiangxi Electric Power Co., LTD. E-mail: qiurixuanwork@163.com.

Hao Luo was born in Oct. 1998. He is a master student at North China Electric Power University. His major research field is information security. E-mail: 13021490107@163.com.

Si-tong Jing works for the PowerChina Jiangxi Electric Power Engineering Co., LTD. E-mail: aurrucy@126.com.

Xin-xiu Li was born in 2000.She is a master student at North China Electric Power University. Her major research field is information security. E-mail: lixinxiu@163.com.

Yuan-cheng Li was born in Aug. 1970. He is a professor and a supervisor of Doctoral student at North China Electric Power University. His major research field is information security and privacy preserving, cryptography and blockchain, artificial intelligence and security. E-mail: ncepua@163.com.