2024 IEEE 11th International Conference on Cyber Security and Cloud Computing (CSCloud) | 979-8-3503-7698-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CSCLOUD62866.2024.00036

Enhancing Cloud-Native Security Through eBPF Technology

Ming Feng
UCloud Technology Co., Ltd.
ming.feng@ucloud.cn

Jia Zhou
UCloud Technology Co., Ltd.
gian.zhou@ucloud.cn

Yi Tang
UCloud Technology Co., Ltd.
eason.tang@ucloud.cn

***Abstract*—In the cloud industry, eBPF (extended Berkeley Packet Filter) security technology is one of the most popular and influential technologies in the Linux kernel in recent years. With the rapid development of networking and cloud-native technologies, eBPF is extensively applied in network and security, performance analysis, container and cloud-native environments, operations and troubleshooting, as well as observability of application systems. This paper focuses on the practical application of eBPF technology in cloud environments for ensuring the secure operation, event-driven security handling capability, and performance hotspots observation of transaction systems. The integration of eBPF technology significantly enhances efficiency and reduces costs across the development, testing, and operational phases of systems, providing effective means for security optimization, assisting in testing, and facilitating fault localization and troubleshooting during secure production operations. Moreover, eBPF plays a crucial role in handling security events in cloud environments.**

***Index Terms*—eBPF, Cloud-native security, Agent, UCloud UHIDS, Port Blocking**

I. Introduction

The advancement of modern operating systems has ushered in a significant evolution in the interaction with and augmentation of system capabilities through extended Berkeley Packet Filter (eBPF) technology. Serving as a potent instrument, eBPF empowers the Linux kernel to dynamically respond to events and execute sandboxed programs, thus becoming a fundamental pillar supporting system observability, networking, and security functionalities. However, akin to any technology interfacing closely with the kernel, ensuring the integrity and security of the eBPF runtime is imperative. This paper delves into the often-overlooked security nuances of eBPF and explores mechanisms aimed at fortifying the protection of eBPF itself. Emphasis is placed on elucidating the pivotal role of the eBPF verifier, scrutinizing existing access control paradigms, and exploring prospective enhancements identified through ongoing research endeavors.

We delineate the application of eBPF in security-centric agent environments in Section 2. By amalgamating eBPF technology with security agents, a synergistic fusion is achieved, enabling heightened efficacy in security monitoring and response mechanisms, thereby bolstering overall system resilience. In Section 3, we elucidate the integration of eBPF within the UCloud UHIDS (UCloud Host-based Intrusion Detection System). Leveraging the prowess of eBPF, the UHIDS system facilitates real-time host monitoring and intrusion detection, furnishing comprehensive security fortification tailored for cloud computing environments [1] [2]. In Section 4, we expound upon the utilization of eBPF to effectuate sub-second port blocking. Through the nimbleness and efficiency [3] [4] inherent to eBPF technology, instantaneous port blocking capabilities are realized, affording rapid mitigation of network security threats and enhancing system robustness. By traversing these pragmatic application domains, this paper endeavors to glean insights into the latent potential of eBPF within the security landscape while providing technical underpinning and guidance for the construction of resilient and secure systems.

II. Application of eBPF in Security Scenario Agents

In a comprehensive eBPF application, typically comprised of both user-space and kernel-space components, each plays a distinct role in executing specific tasks and functionalities. The user-space program is responsible for loading BPF bytecode into the kernel, or is responsible for reading the statistical information or event details returned by the kernel, and performing related data processing and control [5]. The BPF bytecode in the kernel is responsible for executing specific events in the kernel. If necessary, the execution results will also be sent to user space through maps or perf-event events. Therefore, in these two aspects, the user-space program can control some parameters and variables of the eBPF program, as well as the mount point before loading the eBPF program. In other words, when dynamically changing security detection and collection items, all the parameters and variables can be quickly, safely and effectively implanted. Typically, user-space eBPF programs can be developed using the libbpf library to control the loading and execution of kernel-space eBPF programs. By transferring all user-space control and data processing logic to the user-space probe module, packaging and distributing eBPF bytecode through an Agent, and internally controlling the entire eBPF program loading and execution within the Agent, we can combine the advantages of both, enabling any eBPF program to possess the following characteristics:

- **Portable** [6]: Enables eBPF tools and applications to be entirely platform-agnostic and portable, eliminating the need for recompilation for cross-platform distribution.

2693-8928/24/$31.00 ©2024 IEEE
DOI 10.1109/CSCloud62866.2024.00036

- **Isolation** [7]: Leveraging the reliability and isolation of the Agent itself ensures that the loading, execution of eBPF programs, and user-space data processing flow are more secure and reliable.
- **Package Management** [8]: Leveraging the ecosystem and toolchain of the Agent to complete the distribution, management, and loading of eBPF programs or tools. Currently, the eBPF program or tool ecosystem may lack a universal package management or plugin management system.
- **Cross-Language Support**: Currently, eBPF programs are developed in various user-space languages (such as Go, Rust, C, C++, Python, etc.), allowing developers from different backgrounds (C, Go, Rust, Java, TypeScript, etc.) to write user-space eBPF programs in their preferred language without the need to learn a new language.
- **Agility**: For large-scale eBPF applications, the Agent can be directly utilized as a plugin extension platform: extension programs can be delivered and reloaded directly from the control plane at runtime. This not only means that everyone can use official and unmodified applications to load custom extensions, but also that any errors in or updates to eBPF programs can be pushed and/or tested at runtime.
- **Lightweight**: It is difficult to avoid additional CPU and memory consumption in some scenarios when the Agent itself controls resources, and the size of the Agent itself is also influenced by business logic. Therefore, eBPF as a service can be implemented, making the loading and execution of eBPF programs more lightweight, fast, and convenient.

This approach enhances the security, reliability, and efficiency of eBPF applications, rendering them more suitable for a variety of cloud-native security scenarios. [9] By extensively integrating eBPF technology within cloud-native environments, we can effectively realize security monitoring and protection for microservices architectures, containerized deployments, and dynamic networking environments [10]. This comprehensive security solution furnishes a heightened level of security assurance for cloud-native applications, aiding in mitigating evolving threats and attacks while ensuring the continuity and stability of business operations.

III. Application of eBPF in UHIDS

UCloud Host-based Intrusion Detection System (UHIDS) is an essential component of UCloud's security defense. UHIDS is designed to protect the security of user hosts, serving as a host-based security detection system that continuously monitors the security of hosts in real-time. [11]It promptly identifies security vulnerabilities on hosts, assists users in understanding the security status of hosts, and provides targeted reinforcement measures. The host intrusion detection system integrates eBPF technology within its Agent, offering coverage for cloud and container-related scenarios. [12] This enables users to identify and detect security vulnerabilities across various environments effectively.

A. Service Architecture

The UHIDS mainly consists of two components: the UHIDS-Server and the UHIDS-Agent. By installing a lightweight Agent on the host and establishing rule-based event linkage with the cloud-based UHIDS-Server, the security of the host is continuously monitored in real-time. Additionally, through the utilization of eBPF technology, relevant events are monitored, analyzed, and corresponding control strategies are implemented, ensuring the security of the host. In Figure 1, we depict a commonly used service architecture of UHIDS. [13]

B. eBPF works in Agent

The operational principle of eBPF within the Agent is illustrated in Figure 2. Initially, administrators compile eBPF kernel code into BPF bytecode using the compiler. The bytecode is then loaded into the kernel through a system call (Syscall bpf()). The verifier validates the loaded bytecode, and upon successful validation, when probes are triggered, eBPF tracing functions are executed within the kernel space. Concurrently, a mapping area (maps) can be allocated within the kernel space for facilitating data exchange between kernel space and user space. User-space applications can utilize maps to retrieve relevant information collected by the kernel tracing code for preliminary statistical analysis. [14] Subsequently, data is transmitted through the Agent Forward module to the Produce module, and ultimately processed within the Kafka cluster for big data analytics. We show the process of eBPF data flow in Figure 3.

Building upon the aforementioned architecture, we further delve into the internal workings of eBPF within the Agent of UCloud HIDS. Figure 4 [15] illustrates schematic diagram depicting probe types for both application and kernel modules as demonstrated by the bpftrace tool.

EBPF probe types [16] primarily consist of four categories, capable of detecting kernel instructions and user applications in both dynamic and static forms. Kernel probes, a dynamic type of probe, can be set on nearly any kernel instruction to insert dynamic markers or interrupts with minimal system overhead. Kernel probes can be further classified into two types: (1) kprobes, which allow inserting BPF programs before the execution of kernel instructions, and (2) kretprobes, which insert BPF programs when kernel instructions have a return value. Tracepoints are static kernel probes, distinct from kprobes in that they are embedded by kernel developers in the kernel through static code. Due to their enhanced security, static probes, also known as tracepoints, are recommended whenever feasible. User space probes, another dynamic type of probe, permit setting dynamic markers in programs running in user space, including two types: (1) uprobes, where the kernel inserts hooks before the execution of specific instructions in user space programs, and (2) uretprobes, similar to kretprobes but applicable to user space programs, attaching BPF programs to instruction return values, allowing access to return values from registers through BPF code. [17] User Static Defined Tracepoints (USDT) are tracepoints statically written into

user space applications during the coding phase, providing a convenient method for inspecting application behavior.

In the context of UHIDS host intrusion detection application, leveraging the aforementioned probe types enables more comprehensive and secure capture of various types of Trojan events during binary Trojan, Rootkit backdoor Trojan, and virus detection. This facilitates the monitoring of malicious processes and files attacking and disrupting system applications through low-level event invocation. Overall, the application of eBPF technology in security event detection and defense revolves around proactive and reactive approaches.

IV. Use eBPF to achieve port blocking within seconds

When detecting and mitigating security incidents in cloud environments, proactive defense prior to intrusion attempts is paramount. Converging asset exposure and mitigating the longstanding challenge of asset vulnerability exposure to the public network have been persistent issues in the cloud security domain. [18]Typically, addressing asset port exposure suffices to mitigate the majority of intrusion-related security risks for enterprises.

Figure 5 illustrates the application architecture of UHIDS's Agent integrated with the business system for port exposure detection, leveraging the distinctive features of eBPF to timely control security risks existing on the host system. [19] Here, XDP (eXpress Data Path) is utilized to provide a high-performance, programmable network data path for the Linux kernel. The primary advantages of XDP include rapid execution and termination of XDP programs without loops, supporting network offloading. [20] Consequently, this enables faster and more secure control over specific ports or even application traffic without impacting other "legitimate" application services, achieving precise protection.

V. Conclusion

In summary, the application of eBPF technology in the field of cloud security not only demonstrates its powerful capabilities in enhancing system performance, network security, and fault troubleshooting, but also achieves efficient processing and real-time monitoring of security events in cloud environments through integration with the Agent. Through the case analysis of the UCloud UHIDS host intrusion detection system, we can observe the effectiveness and reliability of eBPF technology in practical applications. It not only provides fine-grained security detection but also enables rapid response and processing of security events. Furthermore, the portability, isolation, and cross-language capabilities of eBPF technology allow it to better adapt to the ever-changing cloud environment, providing a more flexible and powerful tool for cloud security. Through these practices, eBPF technology not only enhances the security of cloud services but also provides solid technical support for the continuous innovation and development of cloud services.

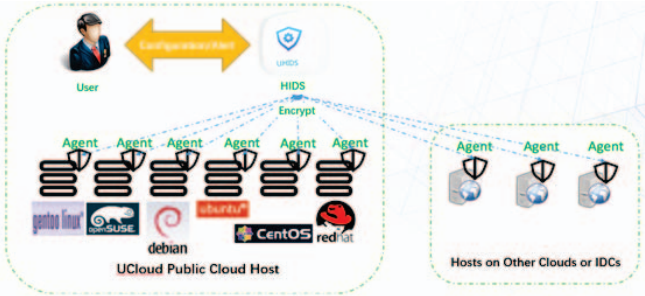

Fig. 1. Service architecture of UHIDS.

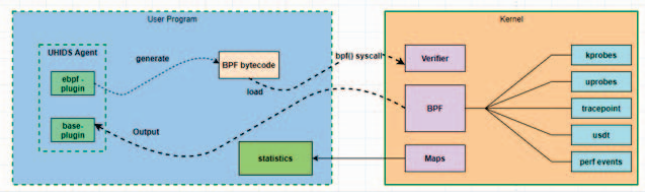

Fig. 2. Principle of eBPF Operation in the Agent.

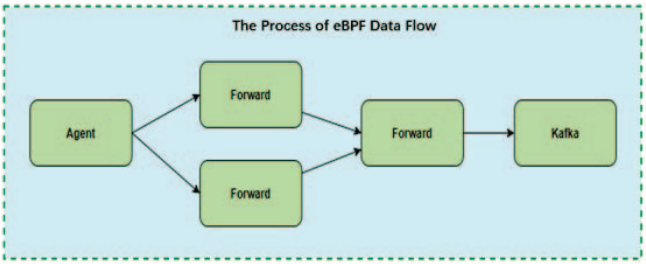

Fig. 3. The process of eBPF data flow.

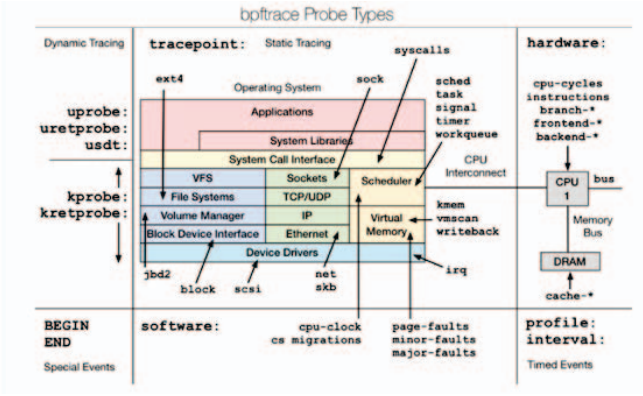

Fig. 4. Schematic diagram of bpftrace probe types for application and kernel modules.

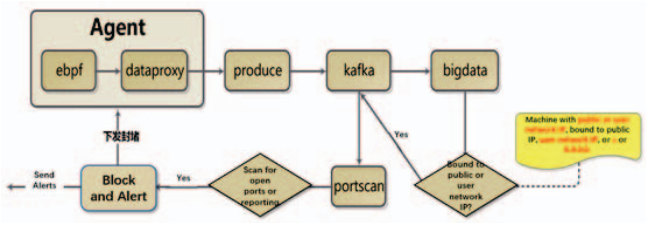

Fig. 5. The application instance architecture of UHIDS's Agent integrated with the business system for port exposure detection.

ACKNOWLEDGMENT

This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFC3302300; in part by the Natural Science Foundation of China under Grant 92046024, Grant 92146002, and Grant 61873309; in part by the Shanghai Science and Technology Project under Grant 22510761000, and Grant 23511100500; and in part by the Intel Sponsored Research Agreement under Grant Intel CG #89533661.

REFERENCES

[1] X. Du, S. Tang, Z. Lu, J. Wet, K. Gai, and P. C. Hung, "A novel data placement strategy for data-sharing scientific workflows in heterogeneous edge-cloud computing environments," in *2020 IEEE International Conference on Web Services (ICWS)*. IEEE, 2020, pp. 498–507.

[2] X. Du, S. Tang, Z. Lu, K. Gai, J. Wu, and P. C. Hung, "Scientific workflows in iot environments: a data placement strategy based on heterogeneous edge-cloud computing," *ACM Transactions on Management Information Systems (TMIS)*, vol. 13, no. 4, pp. 1–26, 2022.

[3] X. Du, X. Chen, Z. Lu, Q. Duan, Y. Wang, and J. Wu, "Biecs: A blockchain-based intelligent edge cooperation system for latency-sensitive services," in *2022 IEEE International Conference on Web Services (ICWS)*. IEEE, 2022, pp. 367–372.

[4] S. Tang, X. Du, Z. Lu, K. Gai, J. Wu, P. C. Hung, and K.-K. R. Choo, "Coordinate-based efficient indexing mechanism for intelligent iot systems in heterogeneous edge computing," *Journal of Parallel and Distributed Computing*, vol. 166, pp. 45–56, 2022.

[5] X. Du, S. Tang, Z. Lu, K. Gai, J. Wu, and P. C. Hung, "Scientific workflows in iot environments: a data placement strategy based on heterogeneous edge-cloud computing," *ACM Transactions on Management Information Systems (TMIS)*, vol. 13, no. 4, pp. 1–26, 2022.

[6] Q. Zeng, M. Kavousi, Y. Luo, L. Jin, and Y. Chen, "Full-stack vulnerability analysis of the cloud-native platform," *Computers & Security*, vol. 129, p. 103173, 2023.

[7] J. Nam, S. Lee, P. Porras, V. Yegneswaran, and S. Shin, "Secure inter-container communications using xdp/ebpf," *IEEE/ACM Transactions on Networking*, vol. 31, no. 2, pp. 934–947, 2022.

[8] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[9] X. Du, Y. Liu, Z. Lu, Q. Duan, J. Feng, J. Wu, B. Chen, and Q. Zheng, "A low-latency communication design for brain simulations," *IEEE Network*, vol. 36, no. 2, pp. 8–15, 2022.

[10] M. R. S. Sedghpour and P. Townend, "Service mesh and ebpf-powered microservices: A survey and future directions," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2022, pp. 176–184.

[11] G. Fournier, S. Afchain, and S. Baubeau, "Runtime security monitoring with ebpf," in *17th SSTIC Symposium sur la Sécurité des Technologies de l'Information et de la Communication*, 2021.

[12] R. Deng, X. Du, Z. Lu, Q. Duan, S.-C. Huang, and J. Wu, "Hsfl: Efficient and privacy-preserving offloading for split and federated learning in iot services," in *2023 IEEE International Conference on Web Services (ICWS)*. IEEE, 2023, pp. 658–668.

[13] R. Gassais, N. Ezzati-Jivan, J. M. Fernandez, D. Aloise, and M. R. Dagenais, "Multi-level host-based intrusion detection system for internet of things," *Journal of Cloud Computing*, vol. 9, no. 1, p. 62, 2020.

[14] L. Wüstrich, M. Schacherbauer, M. Budeus, D. Freiherr von Künßberg, S. Gallenmüller, M.-O. Pahl, and G. Carle, "Network profiles for detecting application-characteristic behavior using linux ebpf," in *Proceedings of the 1st Workshop on eBPF and Kernel Extensions*, 2023, pp. 8–14.

[15] B. Gregg, "Linux performance analysis new tools and old secrets," in *Usenix Lisa 2014 conference*, 2014.

[16] J. Levin and T. A. Benson, "Viperprobe: Rethinking microservice observability with ebpf," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE, 2020, pp. 1–8.

[17] P. Chen, X. Du, Z. Lu, J. Wu, and P. C. Hung, "Evfl: An explainable vertical federated learning for data-oriented artificial intelligence systems," *Journal of Systems Architecture*, vol. 126, p. 102474, 2022.

[18] S. A. Zadeh, A. Munir, M. M. Bahnasy, S. Ketabi, and Y. Ganjali, "On augmenting tcp/ip stack via ebpf," in *Proceedings of the 1st Workshop on eBPF and Kernel Extensions*, 2023, pp. 15–20.

[19] M. Abranches, O. Michel, E. Keller, and S. Schmid, "Efficient network monitoring applications in the kernel with ebpf and xdp," in *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2021, pp. 28–34.

[20] A. Rivitti, R. Bifulco, A. Tulumello, M. Bonola, and S. Pontarelli, "ehdl: Turning ebpf/xdp programs into hardware designs for the nic," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 208–223.