Anomaly Detection on Cloud Native Systems using eBPF probes

Felix Schäfer - s1147681 - Cyber Security and AI
Supervised by: Prof. Lejla Batina, Zhuoran Liu
Start and End Date: 01.02.2026 - 01.08.2026

Introduction

Background and Motivation

Cloud native systems have become the de facto standard for deploying modern applications, offering scalability, resilience, and operational efficiency. However, the distributed and dynamic nature of these environments introduces significant security and operational challenges. Traditional monitoring approaches that rely on application-level logging or external probes often miss low-level system events that could indicate security breaches, performance degradation, or operational anomalies. Extended Berkeley Packet Filter (eBPF) technology enables safe, efficient instrumentation of the Linux kernel without modifying kernel source code or loading kernel modules. This capability makes eBPF particularly valuable for monitoring cloud native systems, where understanding system-level behavior is crucial for both security and performance analysis. By hooking into kernel events such as file system operations, network activity, and process execution, eBPF probes can capture a comprehensive view of system behavior with minimal overhead.

The challenge now lies not only in collecting this data but in making sense of it. The volume of events generated by cloud native systems can be overwhelming, and distinguishing normal operational patterns from anomalous behavior requires sophisticated analysis techniques. Anomaly detection in this context must operate in real-time and adapt to changing workloads. Today, practitioners already use Falco, a widely deployed rule-based runtime detection system for Kubernetes, and this project aims to build on that foundation by extending rule-based detection with anomaly detection to reduce reliance on manually crafted rules [1]. This project addresses the gap between low-level system observability and actionable security insights by developing an integrated system that combines eBPF-based event collection with online anomaly detection models. Rather than aiming for a low false positive rate suitable for automated alerting, this system is designed to provide forensic hints and indicators for later human analysis. The system is expected to produce a significant number of false positives, which is acceptable given its role as a forensic tool rather than an alerting system. The goal is to create a practical, deployable solution that can operate continuously in production environments, flagging potentially interesting events that warrant further investigation by security teams, rather than generating high-confidence alerts.

Short Description

This project develops and validates a two-component system that records predefined triggers with eBPF probes and applies an AI model to detect anomalous behavior in those recordings. The system will be deployed in a realistic cloud native environment and evaluated on real or

carefully constructed datasets to measure detection quality and operational feasibility. The goal is not to reach state-of-the-art accuracy or to minimize false positives, but to deliver a practical detector that provides forensic indicators for later human analysis. The system is expected to produce many false positives, which is acceptable as it serves as a forensic tool rather than an alerting system. The expected outcome is a reproducible pipeline from probe design and data collection to model training, evaluation, and deployment guidance.

Research Questions

- Can we use eBPF probes to extract meaningful information from cloud native systems?
- Which AI methods are suitable for anomaly detection based on the collected data and constraints of online monitoring?
- What detection accuracy and false positive rate can the system achieve under realistic workloads?
- How does the trade-off between model complexity and operational overhead affect practical deployment?

Methods

- **Data Collection:** Define probe points and collect a dataset of cloud native system logs and events via eBPF probes. Acquire data from an industry partner if possible, otherwise construct a realistic synthetic workload that includes normal and anomalous behavior.
- **Data Processing:** Normalize, enrich, and structure the collected data into features suitable for model training and evaluation, including temporal aggregation and context features where relevant.
- **Model Training:** Train one or more anomaly detection models and document the effect of feature choices, model family, and training strategy on detection quality.
- **Model Evaluation:** Evaluate the model using precision, recall, false positive rate, latency, and resource overhead. Compare baseline models to justify the chosen approach.

State of the Art

eBPF Technology

eBPF is a technology that allows writing code that runs in the kernel of the operating system with safety guarantees through a verifier. This enables efficient, low-overhead instrumentation of kernel events without the risks associated with kernel modules. The BCC (BPF Compiler Collection) library provides high-level abstractions for developing eBPF programs in Python and C, making it accessible for building monitoring tools [2]. This project will leverage BCC to implement probes that capture file system operations and other relevant kernel events in cloud native environments.

Falco and Rule-based Runtime Detection

Falco is a widely used cloud native runtime security system for Kubernetes and containers that detects suspicious behavior by evaluating kernel event streams against a rule engine [1]. Its detection logic is defined in YAML rules, which require security teams to handcraft precise conditions and maintain them as environments evolve [3]. Falco is the practical origin of this project: we aim to extend the Falco-style rule-based approach with anomaly detection so that operators are not solely dependent on manually curated rules for discovering novel or environment-specific behavior.

eBPF-based Anomaly Detection Prototypes

Existing prototypes show that pairing eBPF syscall tracing with unsupervised learning can surface behavioral anomalies without predefined rules. Evilsocket demonstrates syscall-level process behavior anomaly detection using eBPF and autoencoders, highlighting practical feasibility but also a heavy reliance on handcrafted feature windows and offline training [4], [5]. A similar open implementation uses eBPF syscall streams and a PyTorch autoencoder for anomaly scoring [6]. These systems validate the core idea but remain focused on single-host syscall streams, limited deployment guidance, and non-online learning. This project extends those efforts to cloud native environments, modular online models, and a reproducible pipeline from data capture to streaming detection.

Cloud Native Systems

Cloud native systems are designed to be deployed in cloud environments, typically using container orchestration platforms like Kubernetes. These systems are characterized by their distributed nature, dynamic scaling, and microservices architecture. K3s is a lightweight, certified Kubernetes distribution optimized for resource-constrained environments and edge computing scenarios [7]. This project will use a k3s cluster to provide a realistic deployment environment that balances complexity with practical constraints.

Anomaly Detection in Streaming Data

Iglesias Vázquez et al. [8] conducted a comprehensive comparison and evaluation of eight state-of-the-art streaming anomaly detection algorithms. Their work revealed that algorithm selection should depend on data characteristics such as locality (whether outlierness is relative to local contexts), relativeness (if past data defines outlierness), and concept drift (its intensity and frequency). The study demonstrated that different algorithms excel under different conditions, and that understanding these characteristics in advance is crucial for optimal design. This project will build upon these findings by evaluating multiple streaming algorithms in the specific context of eBPF-collected system events, where temporal patterns and concept drift are expected due to varying workloads and system states.

Online Anomaly Detection Models

For lightweight, fully streaming settings, Streaming Half-Space Trees (HS-Trees) provide constant-time updates and fixed memory via randomized partitioning of the feature space [9]. This efficiency makes them attractive for high-rate eBPF streams but raises questions about sensitivity to feature scaling and sparse, high-cardinality inputs. LODA (Lightweight Online Detector of Anomalies) uses ensembles of random projections with online density estimation to achieve fast updates and provide feature-level explanations [10]. It is well suited for vectorized event streams but may underperform when anomalies are highly contextual or localized. MemStream introduces a memory-augmented autoencoder that adapts to concept drift while maintaining a bounded memory footprint, balancing adaptability with model complexity [11]. The project will compare these models under realistic workload drift to understand where their assumptions break and which trade-offs fit kernel-level telemetry.

Log-Based Anomaly Detection

Vervaet's MoniLog [12] introduced a distributed approach for detecting real-time anomalies in large-scale cloud environments by analyzing log streams. MoniLog structures log streams and monitors for sequential and quantitative anomalies, with a classifier that learns from administrator actions to evaluate anomaly criticality. While MoniLog operates on application-level logs, this project advances the field by working at the kernel level with eBPF, capturing events that may not appear in application logs and providing a more comprehensive view of system behavior. The integration of online learning models, similar to MoniLog's adaptive approach, will enable the system to improve over time based on operator feedback.

Natural Language Processing for System Events

BERT (Bidirectional Encoder Representations from Transformers) [13] revolutionized natural language processing by enabling deep bidirectional understanding of text through pre-training on large corpora. In the context of system monitoring, file paths, process names, and other textual system identifiers can be analyzed using NLP techniques. This project will explore whether BERT-based embeddings can improve anomaly detection by capturing semantic relationships in system event data, such as understanding that access to `/etc/passwd` and `/etc/shadow` are related security-sensitive operations, even if they appear as distinct paths.

Online Machine Learning

The River library [14] provides a comprehensive collection of online machine learning algorithms designed for streaming data. Unlike batch learning approaches, online algorithms process data incrementally, making predictions and updating models in real-time without requiring separate training phases. This capability is essential for anomaly detection in production systems, where models must adapt continuously to changing workloads and operational patterns. This project will leverage River's algorithms, particularly those evaluated in the streaming anomaly detection study, to build an online detection system that can operate continuously in production environments.

Advancing the State of the Art

While existing work has explored anomaly detection in streaming data and log analysis, this project uniquely combines eBPF-based kernel-level event collection with online anomaly detection models in a cloud native deployment. The integration of these technologies addresses several gaps: (1) capturing events that may not appear in application logs, (2) providing real-time detection with online learning capabilities, (3) operating efficiently in resource-constrained cloud environments, and (4) delivering a complete, deployable system rather than just algorithmic contributions. The evaluation will focus on practical deployment considerations, including detection accuracy, false positive rates, latency, and resource overhead, to ensure the system is viable for real-world use.

Rough Global Planning

- **Weeks 1-2:** Further literature review, refine research questions, define probe scope and success metrics.
- **Weeks 3-6:** Implement and validate eBPF probes, set up the k3s environment, and secure access to data sources.
- **Weeks 6-12:** Data collection, feature engineering, and baseline model experiments.
- **Weeks 12-17:** Model evaluation, error analysis, and iteration on probes or model design based on findings.
- **Weeks 17-24:** Write the thesis, integrate results, and finalize documentation and reproducibility materials.

References

[1] Falco Project, *Falco documentation*, https://falco.org/docs, Runtime security for cloud native environments, 2024. [Online]. Available: https://falco.org/docs

[2] IOVisor Project, *Bcc - tools for bpf-based linux io analysis, networking, monitoring, and more*, https://github.com/iovisor/bcc, GitHub repository, 2024. [Online]. Available: https://github.com/iovisor/bcc

[3] Falco Project, *Falco rules*, https://falco.org/docs/rules/, Rule language and configuration, 2024. [Online]. Available: https://falco.org/docs/rules/

[4] S. Margaritelli, *Process behaviour anomaly detection using ebpf and unsupervised learning autoencoders*, https://www.evilsocket.net/2022/08/15/Process-behaviour-anomaly-detection-using-eBPF-and-unsupervised-learning-Autoencoders/, Blog post, 2022. [Online]. Available: https://www.evilsocket.net/2022/08/15/Process-behaviour-anomaly-detection-using-eBPF-and-unsupervised-learning-Autoencoders/

[5] S. Margaritelli, *Ebpf-process-anomaly-detection*, https://github.com/evilsocket/ebpf-process-anomaly-detection, GitHub repository, 2022. [Online]. Available: https://github.com/evilsocket/ebpf-process-anomaly-detection

[6] exploringweirdmachines, *Ebpf-syscall-anomaly-detection*, https://github.com/exploringweirdmachines/ebpf-syscall-anomaly-detection, GitHub repository, 2022. [Online]. Available: https://github.com/exploringweirdmachines/ebpf-syscall-anomaly-detection

[7] K3s Project, *K3s - lightweight kubernetes*, https://k3s.io/, The certified Kubernetes distribution built for IoT & Edge computing, 2024. [Online]. Available: https://k3s.io/

[8] F. Iglesias Vázquez, A. Hartl, T. Zseby, and A. Zimek, "Anomaly detection in streaming data: A comparison and evaluation study," *Expert Systems with Applications*, vol. 233, p. 120 994, 2023, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.120994 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423014963

[9] S. Tan, K. M. Ting, and T. F. Liu, "Fast anomaly detection for streaming data," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, 2011. [Online]. Available: https://www.ijcai.org/Proceedings/11/Papers/254.pdf

[10] J. Pevny, "Loda: Lightweight on-line detector of anomalies," *Machine Learning*, 2016. DOI: 10.1007/s10994-015-5521-0 [Online]. Available: https://link.springer.com/article/10.1007/s10994-015-5521-0

[11] S. Bhatia, A. Jain, S. Srivastava, K. Kawaguchi, and B. Hooi, "Memstream: Memory-based streaming anomaly detection," in *Proceedings of the ACM Web Conference 2022*, 2022. DOI: 10.1145/3485447.3512221 [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3485447.3512221

[12] A. Vervaet, "Monilog: An automated log-based anomaly detection system for cloud computing infrastructures," *arXiv preprint*, Oct. 2023. arXiv: 2304.11940 [cs.LG]. [Online]. Available: https://arxiv.org/pdf/2304.11940

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint*, Oct. 2018. arXiv: 1810.04805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805

[14] J. Montiel et al., "River: Machine learning for streaming data in python," 2021. [Online]. Available: https://www.jmlr.org/papers/volume22/20-1380/20-1380.pdf