

A Real-Time eBPF-Based Intrusion Detection Framework for Adaptive and Scalable Linux Kernel Security

Maryam Alaa Zaki¹, Nuha Omran Abokhdair²

¹ Department of Computer Engineering and Sciences, Faculty of Science, University of Zawia, Libya,

² Department of Computer Science, Faculty of Science, University of Zawia, Libya

*Corresponding author email: abo.khdeir@zu.edu.ly

Received: 12/8/2026 Revised: 22/8/2026 Accepted: 3/9/2026 Publication: 20/9/2026

ABSTRACT

The Linux kernel represents a critical attack surface due to its privileged execution level, making it a frequent target for attacks such as privilege escalation and advanced persistent threats (APTs). Traditional intrusion detection systems (IDS) typically operate in user space and rely on static or signature based techniques, limiting visibility into kernel-level activities. This paper presents a real-time intrusion detection framework based on Extended Berkeley Packet Filter (eBPF). The framework utilizes eBPF programs attached to kernel trace points and kprobes to capture runtime events with low overhead, enabling fine-grained system observability without requiring kernel modification or system restart.

The proposed approach adopts behavior-based detection using contextual runtime features and a Random Forest classifier for runtime classification. Dynamic policy adaptation is supported through eBPF maps. The framework was evaluated using a controlled dataset containing approximately 38,000 kernel runtime events collected from benign activities and controlled attack scenarios.

Experimental results achieved 99.92% accuracy and 0.99 recall while maintaining CPU utilization below 1% during continuous monitoring. Additional evaluation using the public DongTing benchmark dataset achieved 99.60% accuracy, demonstrating strong generalization capability across heterogeneous kernel behavior environments.

The results indicate that combining eBPF monitoring with lightweight machine learning provides an effective and scalable approach for kernel-level intrusion detection in Linux systems.

Keywords: eBPF, Linux Kernel Security, Intrusion Detection System (IDS), Machine Learning, Runtime Monitoring.

إطار عمل لكشف الاختراقات اللحظية المعتمد على eBPF من أجل حماية تكيفية وقابلة للتوسع لنواة لينكس.

مريم علاء زكي¹، نهى عمران ابوخضير²

¹ القسم هندسة علوم الحاسوب، كلية العلوم، جامعة الزاوية، ليبيا

² القسم علوم الحاسوب، كلية العلوم، جامعة الزاوية، ليبيا

ملخص البحث

تمثل نواة لينكس (Linux Kernel) سطح هجوم حساساً للغاية بسبب مستوى الصلاحيات المرتفع الذي تعمل به، مما يجعلها هدفاً شائعاً للهجمات مثل تصعيد الصلاحيات (Privilege Escalation) والتهديدات المستمرة المتقدمة (APTs). تعمل أنظمة كشف الاختراق التقليدية (IDS) غالباً داخل مساحة المستخدم (User Space) وتعتمد على تقنيات ثابتة أو قائمة على التوقيعات (Signature-Based)، مما يحد من قدرتها على رؤية الأنشطة التي تحدث داخل النواة (Kernel-Level Activities). تقدم هذه الورقة إطار عمل لكشف الاختراقات اللحظية يعتمد على تقنية Extended Berkeley Packet Filter (eBPF). النظام يربط eBPF ببرامج مرتبطة بـ kernel tracepoints و kprobes لالتقاط الأحداث أثناء التشغيل (Runtime Events) مع استهلاك منخفض للموارد، مما يتيح مراقبة دقيقة للنظام دون الحاجة إلى تعديل النواة أو إعادة تشغيل النظام. يعتمد النظام المقترح على أسلوب كشف قائم على تحليل السلوك (Behavior-Based Detection) باستخدام خصائص سياقية مستخرجة من بيئة التشغيل، بالإضافة إلى استخدام نموذج Random Forest لتصنيف الأنشطة أثناء التشغيل. كما يدعم النظام التكيف الديناميكي للسياسات (Dynamic Policy Adaptation) باستخدام eBPF maps. تم تقييم النظام باستخدام مجموعة بيانات خاصة تحتوي على حوالي 38,000 حدث تشغيل داخل النواة، تم جمعها من أنشطة طبيعية وسيناريوهات هجومية محكمة. أظهرت النتائج التجريبية دقة بلغت 99.92% وقيمة Recall تساوي 0.99، مع المحافظة على استهلاك للمعالج أقل من 1% أثناء المراقبة المستمرة.

كما تم إجراء تقييم إضافي باستخدام مجموعة البيانات العامة DongTing benchmark dataset، حيث حقق النظام دقة بلغت 99.60%، مما يدل على قدرة جيدة على التعميم عبر بيانات سلوكية مختلفة داخل النواة. تشير النتائج إلى أن دمج المراقبة المعتمدة على eBPF مع تقنيات تعلم الآلة الخفيفة يوفر نهجًا فعالًا وقابلًا للتوسع لكشف الاختراقات على مستوى نواة لينكس. الكلمات الدالة: eBPF، أمن نواة لينكس، أنظمة كشف الاختراق، تعلم الآلة، المراقبة أثناء التشغيل.

1. Introduction

1.1 Context

The Linux kernel is the core component of modern operating systems, responsible for resource management, hardware abstraction, and enforcement of security policies. Due to its privileged execution level, it represents a critical attack surface and is frequently targeted by attacks such as privilege escalation, rootkits, and advanced persistent threats (APTs) [1–3].

Kernel-level attacks have become increasingly sophisticated, often operating with high stealth and persistence.

By executing within or below the kernel layer, such attacks can bypass traditional security mechanisms, making kernel security a major challenge in modern system protection [1, 4]. Most intrusion detection systems (IDS) operate in user space and rely primarily on static or signature-based techniques. Although effective against known attack patterns, these approaches suffer from limited visibility into kernel-level activities, increased detection latency, and reduced adaptability to previously unseen threats [5, 6].

In addition, user-space monitoring introduces performance overhead due to frequent context switching between kernel and user space. Extended Berkeley Packet Filter (eBPF) provides a practical mechanism for addressing these limitations by enabling safe and efficient execution of monitoring programs directly within the Linux kernel. By attaching programs to trace points and kprobes, eBPF enables real-time observation of system calls and runtime events with minimal overhead, as demonstrated by Vieira et al. [7] and Høiland-Jørgensen et al. [8].

Recent advances in eBPF-based runtime monitoring and machine learning integration demonstrate the potential of combining kernel-level observability with adaptive threat detection [3, 9, 10]. However, efficiently processing high-volume kernel events while maintaining scalability, low-latency, and minimal runtime overhead remains a significant challenge [11].

1.2 Problem Statement

Despite recent advances in runtime monitoring and kernel observability, existing solutions remain limited in providing adaptive and real-time intrusion detection at the kernel-level. Many IDS frameworks operate primarily in user space, while kernel-level security solutions often require intrusive modifications or system restarts, limiting practical deployment and scalability.

eBPF enables flexible in-kernel monitoring without requiring kernel modification or reboot [7, 12]. However, most current eBPF-based systems primarily focus on observability and static rule enforcement rather than adaptive behavioral analysis [13, 14].

In practice, many existing systems rely on predefined rules or static policies, limiting their ability to detect evolving or previously unseen attacks [1, 2]. Furthermore, the capability of eBPF maps to support dynamic interaction between kernel-space and user-space analysis components remains underutilized for adaptive policy management and intelligent runtime response.

1.3 Contributions

This work makes the following contributions:

- **Lightweight eBPF-Based Monitoring:** A kernel-level monitoring framework using eBPF programs attached to tracepoints and kprobes to capture runtime system events with minimal overhead [2, 7].
- **Context-Aware Data Collection:** Enrichment of captured events with contextual runtime attributes, including process hierarchy (PID/PPID) and structural file-access information, enabling more accurate behavioral analysis.
- **Behavior-Based Detection:** A lightweight machine learning approach that models relationships between runtime system events to identify anomalous behavioral patterns [5, 9].
- **Dynamic Policy Adaptation:** Utilization of eBPF maps to support runtime updates of monitoring policies without interrupting system execution.

- **Modular Hybrid Architecture:** A scalable architecture that separates kernel-level event collection from user-space behavioral analysis, enabling efficient processing of high-volume event streams while maintaining low-latency and system stability.
- **External Benchmark Validation:** Evaluation of the proposed framework using both a controlled runtime dataset and the publicly available DongTing benchmark dataset to assess classification performance and generalization capability.

Unlike existing eBPF-based security frameworks, which primarily focus on runtime monitoring or static rule enforcement, the proposed framework combines kernel-level event collection, contextual behavioral analysis, lightweight machine learning, dynamic policy adaptation using eBPF maps, and external benchmark validation within a unified architecture. This integrated design improves adaptability, scalability, runtime responsiveness, and generalization capability while maintaining low runtime overhead.

2. Related works

2.1 eBPF-Based Intrusion Detection Systems

Extended Berkeley Packet Filter (eBPF) has emerged as a powerful mechanism for kernel-level monitoring, enabling the safe execution of user-defined programs within the Linux kernel. By attaching programs to trace points and kprobes, eBPF provides fine-grained visibility into system calls and runtime events with minimal overhead, as demonstrated by Vieira et al. [7] and Høiland-Jørgensen et al. [8]. Several modern systems leverage eBPF for runtime security monitoring and observability. Tools such as Falco and Tracee demonstrate the practicality of eBPF for real-time monitoring in production environments [13, 14]. These systems are effective for detecting predefined behavioral patterns with relatively low performance impact. However, most existing approaches rely primarily on static rules or predefined policies, limiting their ability to detect evolving or previously unseen threats [5, 6]. Recent research has explored the integration of machine learning with eBPF to improve adaptive threat detection capabilities. Existing studies demonstrate that combining lightweight machine learning models with kernel-level observability can enhance runtime security while maintaining acceptable performance overhead [3, 9, 10].

2.2 Kernel-Level IDS/IPS Systems

Several kernel-level intrusion detection and prevention systems have been proposed using eBPF. The iKern framework integrates intrusion detection and prevention directly within the kernel, enabling real-time monitoring and mitigation with low overhead [2]. Similarly, recent approaches employ neural networks and lightweight classifiers to improve runtime threat detection and behavioral analysis [10, 11].

Despite these advances, many existing systems remain constrained by static or semi-static detection logic. The limited use of adaptive behavioural modeling reduces their ability to generalize across diverse attack scenarios and dynamic runtime environments [3, 6].

2.3 Comparative Analysis and Limitations

Existing approaches reveal a trade-off between efficiency and adaptability. Rule-based systems such as Falco and Tracee provide efficient real-time monitoring but depend heavily on predefined signatures and static policies [13, 14]. In contrast, machine learning-based approaches improve adaptability but often introduce additional computational complexity and scalability challenges [5, 10]. Several limitations remain common across existing solutions:

- Dependence on static or predefined detection rules, reducing effectiveness against unknown or evolving threats.
- Limited utilization of contextual behavioural features for modeling runtime system activity [5, 6].
- Insufficient support for dynamic runtime policy adaptation and feedback-driven response mechanisms.
- Challenges in processing high-volume kernel events while maintaining low performance overhead and scalability [4, 15].

Furthermore, integrating machine learning with kernel-level monitoring introduces additional challenges related to feature extraction, event stream processing, and efficient kernel-to-user-space communication [4, 9]. Lightweight approaches such as Random Forest and decision tree-based models have demonstrated effectiveness in balancing detection performance and computational cost [3, 15].

2.4 Research Gap and Motivation

Despite recent advances, no existing solution fully combines efficient kernel-level monitoring, behaviour-driven detection, lightweight machine learning, and dynamic policy adaptation within a unified framework. In particular, the runtime capabilities of eBPF maps remain underutilized for adaptive security policy management, despite their ability to support efficient interaction between kernel and user-space components [12].

To address these limitations, this work proposes a lightweight eBPF-based intrusion detection framework that integrates kernel-level monitoring with behaviour-based analysis and supports dynamic policy updates during runtime. The proposed framework aims to improve adaptability, scalability, and real time responsiveness while maintaining low runtime overhead.

3. System Architecture Overview

The proposed system adopts a hybrid architecture that integrates kernel-space monitoring with user-space analysis to enable efficient, scalable, and real-time intrusion detection. The design separates low-level event collection from high-level behavioural analysis, improving modularity, extensibility, and runtime efficiency. The architecture consists of two primary layers: kernel-space and user space. In kernel-space, custom eBPF programs are attached to blocking or mitigation of suspicious activities. By leveraging system call trace points and kprobes to capture runtime system events with high fidelity. These programs operate within a sandboxed execution environment enforced by the eBPF verifier,

ensuring memory safety and controlled execution while maintaining low performance overhead, as demonstrated by Vieira et al. [7] and Høiland-Jørgensen et al. [8].

The monitoring layer captures events related to process execution, file access, and associated system call parameters. Captured events are transferred to user space using a high performance ring buffer mechanism, enabling low-latency and non-blocking communication between kernel and user-space components. Efficient event handling is essential for maintaining scalability under high event throughput, as highlighted in recent eBPF-based runtime monitoring studies [4, 9].

In user space, a data collection module receives the event stream and performs filtering and preprocessing to remove noisy or redundant activities, including repetitive system calls and background processes. This stage reduces data redundancy and improves the consistency of subsequent behavioral analysis. The processed data is forwarded to a behavioral analysis component, where relationships between runtime events are modeled using contextual features such as process hierarchy (PID/PPID), execution context (UID), and file-access patterns. This enables the system to analyze behavioral sequences rather than isolated events, improving anomaly detection capability [5, 6].

A detection module evaluates the extracted behavioral features and classifies system activities as benign or suspicious. A lightweight machine learning model is integrated as a supporting classification component, while the core detection logic remains behavior-driven and grounded in kernel-level observability. Experimental observations indicate that detection performance is strongly influenced by feature quality and dataset balance, particularly in large-scale behavioral datasets [6].

Based on classification outcomes, a policy decision module determines appropriate responses, including alert generation and runtime policy updates. The modular design facilitates extensibility and enables integration of additional response mechanisms without affecting system stability.

A key feature of the proposed architecture is support for dynamic security policy updates using eBPF maps.

These maps act as shared data structures between kernel-space and user space, enabling real-time modification of monitoring parameters and detection rules without requiring program reload or system restart [10, 14].

This mechanism supports adaptive runtime behavior and improves responsiveness to evolving system activities. The architecture also incorporates a feedback-driven mechanism in which classification outcomes are used to refine detection behavior and update monitoring policies over time.

This adaptive approach improves runtime responsiveness and supports continuous refinement of monitoring strategies. Furthermore, the policy decision module can be extended to interact with kernel-level enforcement points, enabling selective blocking or mitigation of suspicious activities. By leveraging eBPF helper functions and runtime-configurable maps, the framework can evolve from a passive intrusion detection system (IDS) into an active intrusion prevention system (IPS).

A Real-Time eBPF-Based Intrusion Detection Framework for Adaptive and Scalable Linux Kernel Security

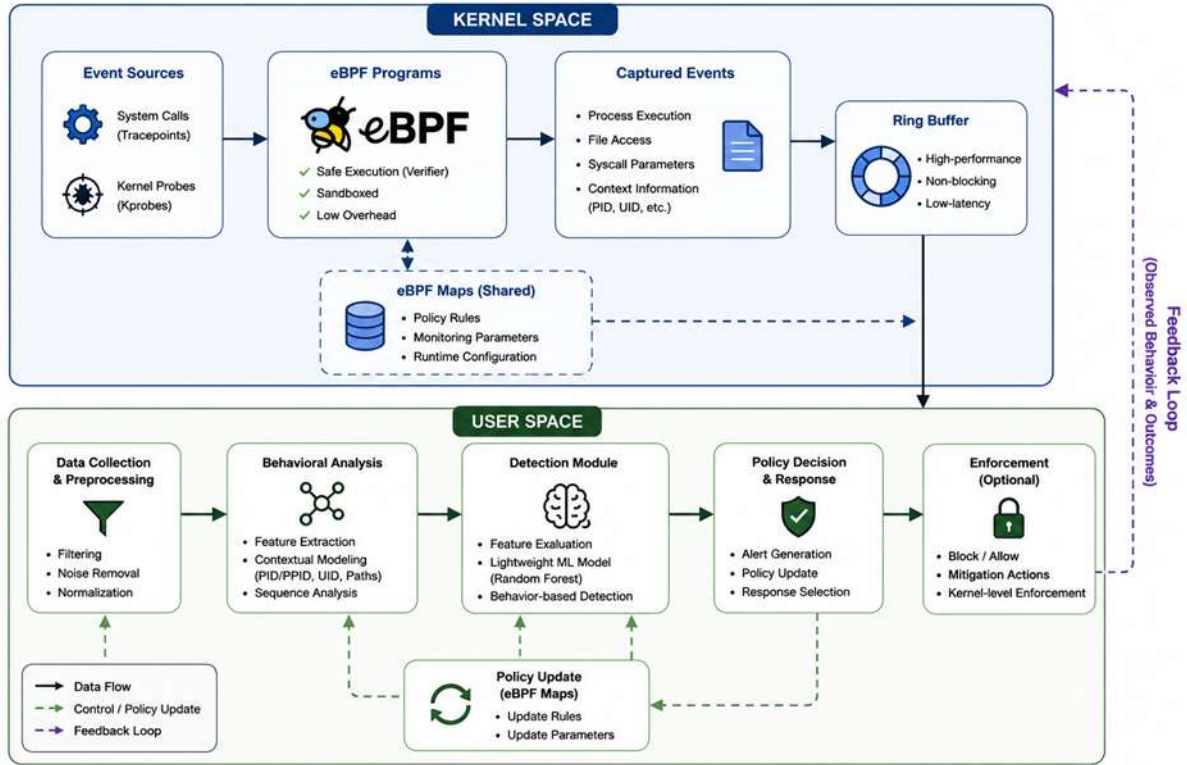


Figure 1. Proposed eBPF-based intrusion detection system architecture

As illustrated in Fig. 1, the proposed architecture enables continuous monitoring and analysis of kernel-level activities while maintaining low runtime overhead and high scalability.

The separation between kernel-space event collection and user space behavioral analysis improves both system safety and extensibility, making the framework suitable for real-world deployment.

4. METHODOLOGY AND SYSTEM DESIGN

4.1 System Overview

The proposed system is an eBPF-based intrusion detection framework that operates at the Linux kernel-level, enabling real time monitoring of system behavior with low runtime overhead.

Runtime events are captured using eBPF programs attached to kernel hooks such as tracepoints and kprobes, enabling direct observation of system calls and process activities. The framework adopts a hybrid architecture in which kernel-level event collection is separated from user-space behavioral analysis. Captured events are transferred efficiently through a high-performance ring buffer mechanism, ensuring low-latency communication between kernel and user-space components, as demonstrated by Vieira et al. [7].

Machine learning is integrated as a lightweight supporting component for classification, while the core framework focuses on behavior-driven analysis and adaptive runtime monitoring. Dynamic policy management is achieved through eBPF maps, enabling real-time updates of monitoring rules without requiring kernel modification or system restart, similar to mechanisms employed in modern eBPF-based monitoring systems [10, 13].

4.2 Threat Model

The proposed framework assumes attackers exploit system vulnerabilities to perform activities such as privilege escalation, unauthorized file access, or process manipulation, as discussed in recent eBPF-based security studies [2, 3].

These activities typically manifest as abnormal syscall sequences, irregular process hierarchies, or suspicious interactions with sensitive system resources.

The framework focuses on behavior-based anomaly detection rather than predefined signature matching. It assumes attackers operate through user-space processes without the ability to compromise the Linux kernel or bypass the eBPF verifier.

The trusted computing base (TCB) includes the Linux kernel, the eBPF verifier, and user-space analysis components, which collectively ensure safe and controlled execution of eBPF programs [7].

Under these assumptions, the framework aims to detect both known and previously unseen threats by identifying deviations from normal runtime behavior while maintaining low performance overhead [16].

4.3 System Architecture

The proposed framework adopts a layered architecture that integrates kernel-space monitoring with user-space behavioral analysis. The architecture consists of a kernel monitoring layer, an event transfer layer based on ring buffer communication, and a user-space analysis layer responsible for preprocessing, feature extraction, behavioral classification, and policy decision making. The kernel-space monitoring layer utilizes eBPF programs attached to tracepoints and kprobes to capture runtime events with low overhead, consistent with modern eBPF-based monitoring approaches [7, 8].

Runtime events are transferred efficiently to user space through ring buffer communication mechanisms, enabling scalable and low-latency event processing [4, 9].

This layered design enables separation between low-level event collection and high-level behavioral analysis, improving modularity, scalability, and runtime efficiency.

The detailed architectural design and component interactions are presented in Section III.

4.4 Data Acquisition and Dataset Construction

To evaluate the robustness and generalization capability of the proposed framework, the experimental evaluation utilized two complementary datasets: a controlled runtime behavioral dataset constructed using the proposed monitoring framework and a publicly available benchmark dataset.

- **Controlled Runtime Dataset:** The primary dataset was constructed using a dual-phase execution strategy consisting of benign runtime activities and controlled adversarial scenarios. Approximately 35,000 benign runtime samples were collected from routine Linux system activities representing normal behavioral variance. In addition, approximately 3,000 suspicious samples were generated through controlled attack-oriented activities, including unauthorized access attempts to sensitive files, reconnaissance operations, and suspicious execution behavior from volatile directories. The resulting dataset contains approximately 38,000 kernel runtime events and provides a behavioral foundation for evaluating runtime anomaly detection performance.

- **Public Benchmark Dataset:** Additional experiments were conducted using the publicly available DongTing kernel behavioral dataset [6] to further evaluate the generalization capability of the proposed framework. The DongTing dataset contains large-scale syscall traces collected from both benign and malicious kernel activities and is widely used in kernel anomaly detection research. The combination of controlled runtime traces and publicly available benchmark data improves the robustness, credibility, and reproducibility of the experimental evaluation. Previous studies have shown that large-scale behavioral datasets significantly improve anomaly detection reliability and contextual behavioral analysis in kernel-level intrusion detection systems [6].

4.5 Data Processing and Behavioral Modeling

Kernel events are exported to user space and transformed into structured behavioral data for analysis. To handle high-volume event streams efficiently, a preprocessing stage filters noisy and redundant activities prior to feature extraction. The extracted features include both structural and contextual runtime attributes, including PID, PPID, UID, path depth, and file-path length. These features enable the framework to model behavioral relationships between runtime events rather than treating individual syscalls independently [5].

Behavioral analysis is performed using a machine learning based classification component designed to distinguish benign and suspicious runtime activities while maintaining scalable event processing and low runtime overhead.

4.6 Policy Decision Layer and Feedback Mechanism

The policy decision layer interprets classification outcomes and determines appropriate runtime responses. A key component of this layer is the utilization of eBPF maps, which enable dynamic interaction between kernel-space monitoring programs and user-space analysis components [14].

In the implemented framework, user-space modules dynamically update policy entries, which are immediately applied by kernel-level eBPF programs. A feedback-driven mechanism is incorporated to refine monitoring behavior over time.

Once suspicious activity is detected, the framework updates monitoring policies to increase surveillance of the associated process or runtime behavior. This adaptive mechanism improves responsiveness and supports continuous refinement of detection strategies.

4.7 System Workflow

The proposed framework operates as a continuous pipeline integrating kernel-level monitoring with user-space behavioral analysis. Runtime events are captured using eBPF programs attached to kernel trace points and kprobes, transferred to user space for preprocessing and feature extraction, and subsequently analyzed by the behavioral detection component. Based on classification outcomes, the framework generates alerts and dynamically updates monitoring policies through eBPF maps without interrupting kernel execution. This workflow enables continuous runtime adaptation while maintaining scalable and low-overhead monitoring behavior.

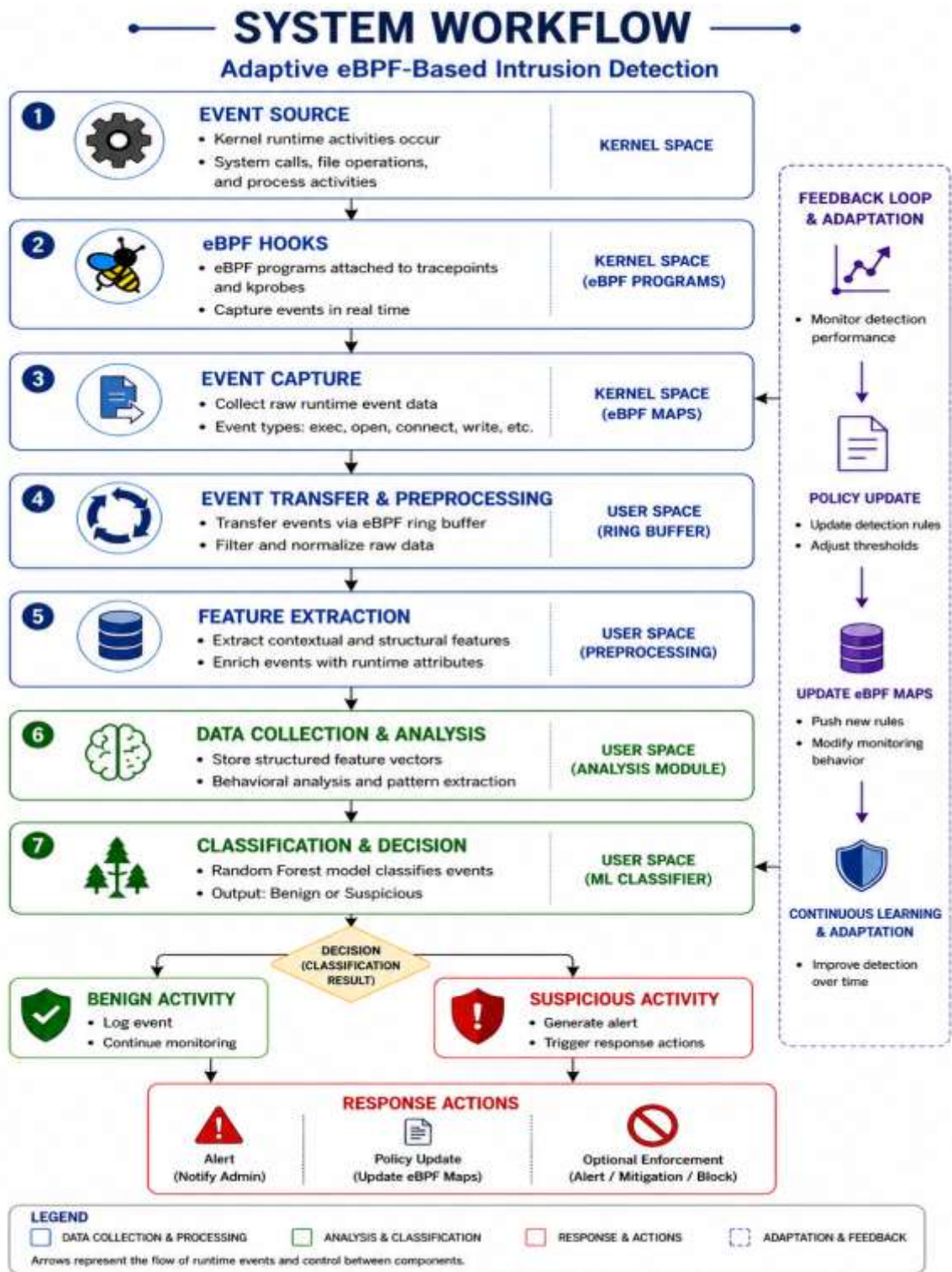


Fig. 2. System workflow for adaptive eBPF-based intrusion detection

4.8 Algorithm Description

Algorithm 1 Adaptive eBPF-Based Intrusion Detection

```

1: while system is running do
2:   Capture event E using eBPF
3:   Transfer E to user space
4:   F  $\xrightarrow{\quad}$  extractFeatures(preprocess(E))
5:   (label, score)  $\xrightarrow{\quad}$  DetectionModel(F)
6:   if label == suspicious then
7:     Trigger alert
8:     Update policy via eBPF maps
9:   end if
10:  Refine detection using feedback
11: end while

```

4.9 Design Objectives

The proposed framework is designed to achieve the following objectives:

- **Efficient Kernel-Level Monitoring:** Direct runtime observation using eBPF programs attached to kernel hooks [7].
- **Low Runtime Overhead:** Minimal performance impact through in-kernel execution and ring buffer communication [8].
- **High Observability:** Fine-grained visibility into runtime system behavior and syscall activity.
- **Behavior-Based Detection:** Identification of anomalous activities through contextual relationships between runtime events [5].
- **Dynamic Policy Adaptation:** Runtime modification of monitoring policies using eBPF maps [14].
- **Scalability and Extensibility:** Deployment without requiring kernel modification or system restart.

5. Implementation

5.1 Development Environment

The proposed framework was implemented in a Linux-based environment to support native eBPF execution and kernel-level tracing capabilities.

eBPF programs were developed using the BCC framework, which enables attachment to kernel trace points and kprobes for efficient runtime monitoring [7, 12].

In user space, Python 3 was used for data collection, preprocessing, behavioral analysis, and machine learning integration. Pandas was utilized for structured data processing and feature manipulation, while Scikit-learn was employed for classification experiments. A Random Forest classifier was selected due to its robustness, stability, and computational efficiency when handling structured behavioral runtime data [3].

Table 1. Implementation Environment

Components	Details
Operating System	Ubuntu 22.04 LTS
Kernel Version	Linux Kernel 6.x
eBPF Framework	BCC
Programming Language	Python 3.12
Machine Learning	Scikit-learn
Data Processing	Pandas
CPU	Intel Core i7 Processor
RAM	GB 16

5.2 eBPF-Based Data Collection

Kernel-level runtime events were collected using custom eBPF programs attached to syscall tracepoints and kprobes. The monitoring layer captures critical runtime activities related to process execution and file access operations, including exec and open system calls [7, 8].

A Real-Time eBPF-Based Intrusion Detection Framework for Adaptive and Scalable Linux Kernel Security

Unlike traditional user-space monitoring approaches, the proposed framework enriches collected events directly within kernel-space by extracting contextual runtime attributes such as PID, PPID, UID, command name, and file path information. This enables behavioral modeling of process relationships and execution chains rather than isolated system events. Captured events are transmitted to user space using a high-performance ring buffer mechanism, providing efficient kernel to-user-space communication with low-latency and minimal overhead. In addition, eBPF maps are utilized to support dynamic runtime interaction between monitoring and analysis components, enabling runtime policy updates without requiring kernel modification or system restart [10, 14].

5.3 Dataset Construction and Preprocessing

The experimental evaluation utilized two complementary datasets: a controlled runtime dataset collected using the proposed eBPF monitoring framework and a publicly available benchmark dataset. The primary dataset was generated from real-time kernel event traces collected under controlled runtime conditions. Each event sample contains contextual runtime attributes including PID, PPID, UID, command name, syscall behavior, and file access information, enabling representation of both structural and contextual system behavior.

Table 2. Correlated runtime dataset summary

Total Samples	38,000+
Benign Samples	35,000
Suspicious Samples	3,000
Data Source	eBPF Runtime Monitoring

Benign samples were generated through routine system activities over a 72-hour period, including software development tasks, package management, command-line interactions, and web browsing. Suspicious samples were generated using controlled adversarial scenarios such as unauthorized access attempts to sensitive files (e.g., /etc/shadow), reconnaissance operations using tools such as nmap, and execution of binaries from volatile directories including /tmp. To further evaluate generalization capability, additional experiments were conducted using the publicly available DongTing kernel behavioral dataset [6], which contains syscall traces collected from both normal and malicious kernel activities. A preprocessing pipeline was applied to improve data quality and reduce runtime noise.

This stage included filtering redundant syscall traces, removing incomplete records, eliminating noisy background activities, and transforming raw runtime events into structured behavioral feature vectors suitable for machine learning analysis.

5.4 Feature Engineering

Feature engineering was designed to capture both structural and contextual characteristics of runtime behavior from eBPF collected kernel events [5, 9].

Categorical runtime attributes, including command names and file paths, were transformed into numerical representations using label-based encoding. Structural features such as PID, PPID, UID, path length, and directory depth were preserved as numerical behavioral attributes due to their discrete runtime nature.

Table 3.Extracted Features

Feature	Description
file_code	Encoded accessed file
command_code	Encoded command name
pid	Process identifier
ppid	Parentprocess identifier
uid	User identifier
path_length	File path length
path_depth	Directory depth
is_sensitive	Access to sensitive paths
is_tmp	Access to temporary directories
is_privileged	Root-level execution

A Real-Time eBPF-Based Intrusion Detection Framework for Adaptive and Scalable Linux Kernel Security

Process hierarchy information obtained through PPID enables modeling of execution chains and parent-child process relationships, while path-related features capture behavioral characteristics associated with file access operations. Sensitive path indicators and privilege-related attributes further improve detection of abnormal runtime activities. The resulting structured feature vectors were used as input for machine learning classification.

5.5 Classification Results

The proposed framework achieved strong classification performance on the controlled runtime dataset. The dataset was divided into training and testing subsets using an 80/20 split ratio.

To ensure reproducibility, a fixed random seed (random_state=42) was applied during dataset splitting. The Random Forest classifier was configured using 100 decision trees (n_estimators=100) with the default Gini impurity criterion and unrestricted tree depth. This configuration provided stable classification performance while maintaining low computational overhead.

All experiments were conducted on the same hardware and software environment to ensure consistency and reproducibility of the evaluation process.

Table 4. Classification performance of the proposed model

Class	Precision	Recall	F1-score	Support
Benign	1.00	1.00	1.00	7147
Suspicious	1.00	0.99	0.99	485
Accuracy	0.9992			

The achieved results indicate high classification capability with very low false positive and false negative rates.

Table 5. Confusion Matrix of the proposed model

	Predicted Benign	Predicted Suspicious
Actual Benign	7146	1
Actual Suspicious	5	480

To further evaluate discriminative capability, ROC analysis was performed.

The resulting ROC curve achieved an AUC score of 0.998, demonstrating strong separation between benign and suspicious runtime activities.

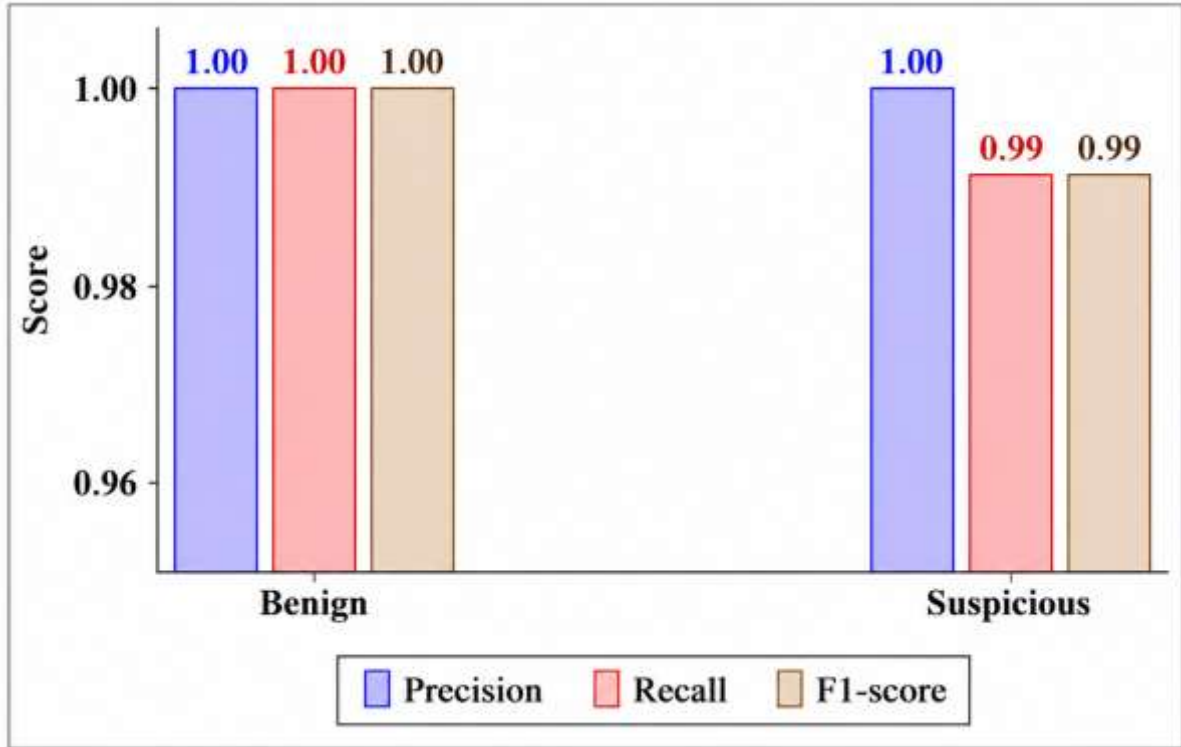


Fig. 3. Precision, recall, and F1-score of the proposed model

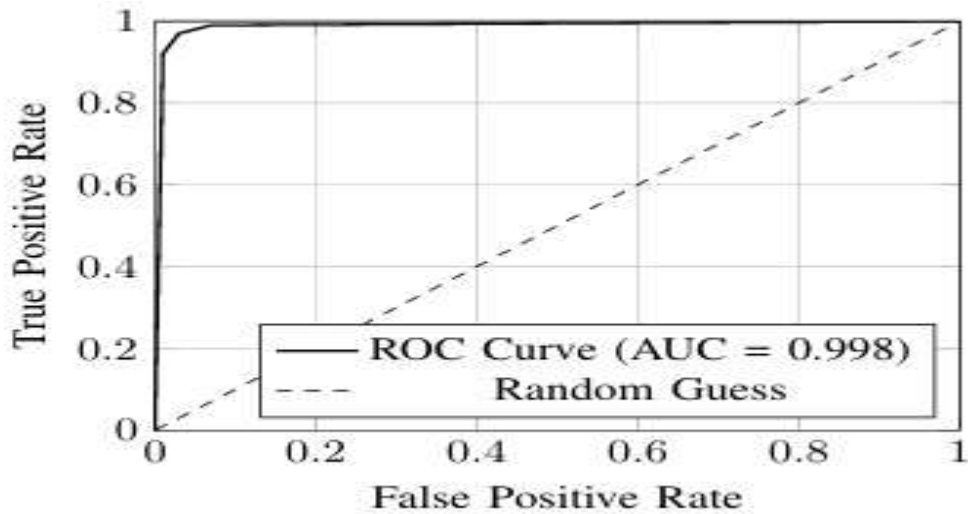


Fig. 4. ROC curve of the proposed Random Forest classifier

To evaluate model stability, 5-fold cross-validation was conducted on the constructed dataset. The Random Forest classifier achieved an average accuracy of 99.81% with low variance across folds, indicating stable and consistent classification behavior.

To further evaluate external generalization capability, additional experiments were conducted using the publicly available DongTing benchmark dataset [6].

The proposed framework achieved an accuracy of 99.60%, demonstrating effective classification performance under benchmark evaluation conditions. The proposed framework successfully distinguishes between benign and suspicious runtime activities in real time. Suspicious interactions involving sensitive system resources are immediately identified and classified using contextual behavioral analysis.

A controlled runtime experiment involving unauthorized access attempts to `/etc/shadow` confirmed the capability of the proposed framework to detect suspicious kernel-level behavior in real time.

```

eBPF Runtime Intrusion Detection
=====

[EVENT] Process Execution
[COMMAND] cat /etc/shadow

[FEATURE EXTRACTION]
PID: 4421
PPID: 3010
UID: 1000
PATH: /etc/shadow
is_sensitive = True

[CLASSIFICATION]
Prediction: SUSPICIOUS

[ALERT]
Unauthorized sensitive file access detected
    
```

Fig. 5. Real-time detection and classification of suspicious runtime activity involving unauthorized access to a sensitive system resource

5.6 Performance Overhead Evaluation

The runtime overhead introduced by the proposed framework was evaluated under normal workload conditions using standard Linux profiling tools [7].

Table 6. System overhead evolution

Metric	Baseline	With Monitoring
CPU Usage (%)	~0.0	~0.5
Memory Usage (MB)	~10	~25
Event Throughput	–	~1000+ events/sec

Although CPU and memory overhead were experimentally evaluated, detection latency and end-to-end response time were not quantitatively measured in the current implementation. These metrics are considered important for assessing real-time deployment performance and will be investigated in future work.

5.7 Discussion

The experimental results demonstrate the effectiveness of combining eBPF-based kernel observability with behavior-based analysis for real-time intrusion detection. Contextual runtime features, particularly process hierarchy information (PPID), sensitive file indicators, and path-related attributes, significantly improved classification performance by modeling relationships between runtime events rather than treating individual system calls independently [5].

To further evaluate the proposed approach, the Random Forest classifier was compared with baseline machine learning models, including Logistic Regression and Decision Tree, as shown in **Table 7**.

Table 7. Performance Comparison with Baseline Machine Learning Models

Model	Accuracy	F1-score	Recall
Logistic Regression	95.84%	0.95	0.94
Decision Tree	97.13%	0.97	0.96
Random Forest (Proposed)	99.92%	0.99	0.99

The Random Forest classifier consistently outperformed the baseline models across all evaluation metrics while maintaining low computational complexity. Its ensemble-based learning strategy proved particularly effective for modeling nonlinear behavioral relationships in structured kernel runtime data.

Although the proposed model achieved high classification accuracy, the possibility of overfitting was carefully considered. To reduce this risk, the model was evaluated using a fixed train-test split, 5-fold cross-validation, and an additional external benchmark dataset (DongTing). The consistent performance across these evaluation strategies indicates that the proposed framework generalizes well beyond the internally constructed dataset.

Feature importance analysis further confirmed the contribution of contextual runtime features, particularly sensitive file access indicators, process hierarchy (PPID), and path-related attributes, to runtime anomaly detection.

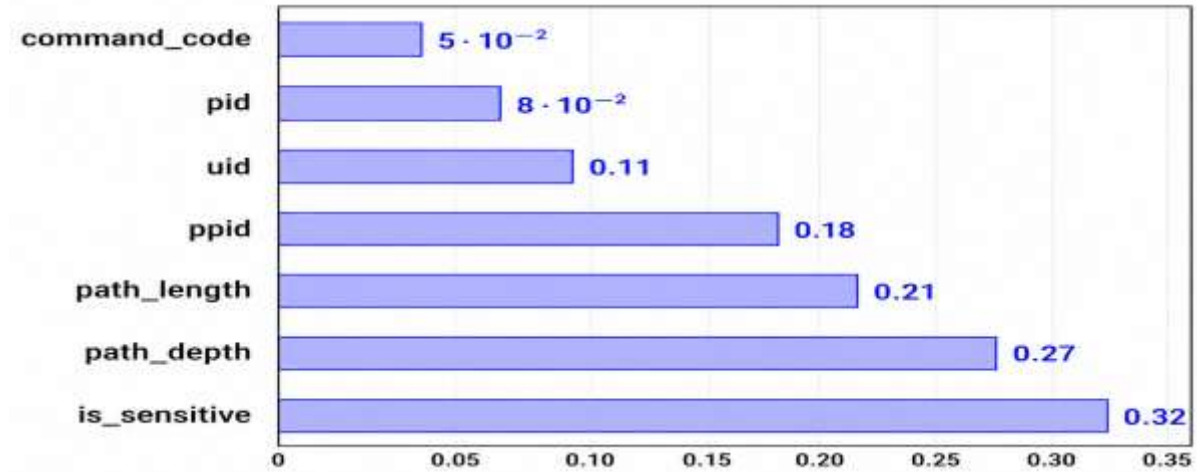


Fig. 6. Feature importance scores extracted from the Random Forest classifier

To further validate external generalization capability, additional experiments were conducted using the publicly available DongTing kernel behavioral dataset [6]. Unlike the internally constructed dataset, DongTing contains syscall traces collected under independent experimental conditions from diverse benign and malicious kernel activities. The proposed framework achieved an accuracy of **99.60%**, demonstrating that the behavioral detection approach generalizes effectively across heterogeneous kernel runtime environments.

The consistently low false-positive and false-negative rates further demonstrate the robustness of contextual behavioral modeling when applied to external kernel runtime traces, improving the credibility and reproducibility of the proposed framework.

Table 8. Comparison with recent EBPF-BASED intrusion detection studies

Study	Dataset Type	ML-Based	Accuracy
Wang et al. [3]	Controlled	No	–
Zhang et al. [11]	Runtime Events	Yes	98.7%
Remya et al. [15]	Network Runtime	Yes	97.9%
Proposed Framework (Controlled)	Kernel Behavioral	Yes	99.92%
Proposed Framework (DongTing)	Public Benchmark	Yes	99.60%

The comparison presented in **Table 8** is intended to provide contextual positioning of the proposed framework relative to representative eBPF-based detection studies. Since the compared studies employ different datasets, experimental environments, and evaluation protocols, the reported accuracies should not be interpreted as direct benchmark comparisons.

A Real-Time eBPF-Based Intrusion Detection Framework for Adaptive and Scalable Linux Kernel Security

Despite the strong experimental results, the current evaluation was conducted under partially controlled runtime conditions. Therefore, additional validation using heterogeneous production environments, long-term workloads, and larger benchmark datasets remains an important direction for future research.

Table 9. Comparison with existing EBPF-based security systems

System	Detection Type	Dynamic Policy	ML-Based	Real-Time
Falco [14]	Rule-based	Limited	No	Yes
Tracee [13]	Rule-based	Limited	No	Yes
iKern [2]	Hybrid	Partial	No	Yes
Proposed Framework	Behavior-Based	Yes	Yes	Yes

Unlike existing eBPF-based security systems, which primarily emphasize runtime monitoring or static rule enforcement, the proposed framework integrates contextual behavioral analysis, lightweight machine learning, adaptive policy management, and external benchmark validation within a unified architecture. This combination improves adaptability while maintaining low runtime overhead and real-time monitoring capability.

6. Conclusion and future work

6.1 Conclusion

This paper presented a lightweight eBPF-based intrusion detection framework for kernel-level monitoring and behavior-driven analysis in Linux environments. The proposed framework leverages in-kernel observability to capture runtime system events with minimal overhead while enabling fine-grained behavioral analysis through contextual feature modeling [5].

By combining eBPF-based monitoring with lightweight machine learning techniques, the system achieves an effective balance between detection accuracy, runtime efficiency, and scalability. Experimental evaluation on both a controlled runtime dataset and the publicly available DongTing benchmark dataset demonstrated strong classification performance. The proposed framework achieved an accuracy of 99.92% on the controlled dataset and 99.60% on the DongTing dataset, indicating strong generalization capability across different kernel behavioral environments. In addition, the use of eBPF maps enables dynamic runtime policy adaptation without requiring kernel modification or system restart, providing a flexible alternative to traditional static rule-based security systems [10, 14].

The experimental results further demonstrate that contextual behavioral features, particularly process hierarchy and file access characteristics, significantly improve runtime anomaly detection capability. Moreover, the proposed framework maintains low runtime overhead, with CPU utilization remaining below 1% during continuous monitoring, confirming the practicality of deploying eBPF-based behavioral intrusion detection in real-world Linux systems.

Although the proposed framework demonstrated strong performance under experimental conditions, additional validation in heterogeneous production environments remains necessary to fully assess long-term robustness, scalability, and deployment reliability.

Overall, the findings indicate that eBPF provides a practical, scalable, and efficient foundation for adaptive intrusion detection and runtime security monitoring at the kernel-level [7].

Compared with existing eBPF-based security frameworks, the proposed approach combines contextual behavioral analysis, lightweight machine learning, adaptive policy management, and external benchmark validation within a unified architecture, providing improved adaptability while maintaining low runtime overhead.

The proposed framework further highlights the potential of combining kernel-level observability with lightweight behavioral analysis for next-generation Linux security systems.

6.2 Future Work

Future work will focus on several directions to further enhance the proposed framework:

- **Extended Benchmark Evaluation:** Expanding experimental evaluation using additional publicly available kernel behavioral datasets beyond DongTing to further assess generalization capability and comparative performance across heterogeneous runtime environments [6].
- **Real-World Deployment:** Evaluating the framework in large-scale production environments to analyze long-term stability, scalability, and runtime robustness under realistic workloads.
- **Advanced Detection Models:** Investigating deep learning based anomaly detection and hybrid behavioral analysis approaches to improve detection of zero-day and stealthy attacks [9, 10].
- **Online Learning Mechanisms:** Implementing adaptive online learning techniques to continuously refine detection behavior according to evolving runtime activity patterns.
- **Intrusion Prevention Capabilities:** Extending the framework toward active intrusion prevention by integrating runtime enforcement mechanisms using eBPF helper functions and dynamic policy controls.
- **Cloud-Native Runtime Security:** Evaluating the framework within containerized and cloud-native infrastructures, including Kubernetes-based environments and distributed runtime systems [3].
- **Multi-Layer Security Integration:** Combining kernel-level behavioral monitoring with network-layer analysis to support unified multi-layer intrusion detection architectures.

7. REFERENCES

- [1] 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, 2022.
- [2] H. J. Hadi, M. Adnan, Y. Cao, F. B. Hussain, N. Ahmad, M. A. Alshara, and Y. Javed, “iKern: Advanced intrusion detection and prevention at the kernel-level using eBPF,” *Technologies*, vol. 12, no. 8, p. 122, 2024.
- [3] S. Remya, M. J. Pillai, B. Niranjana, and P. M. A. Kumar, “eBPF-based runtime detection of semantic DDoS attacks in Linux containers,” *IEEE Access*, vol. 13, pp. 1–15, 2025.
- [4] B. Gbadamosi et al., “The eBPF runtime in the Linux kernel,” *arXiv preprint arXiv:2410.00026*, 2024.
- [5] M. Bachl, J. Fabini, and T. Zseby, “A flow-based IDS using machine learning in eBPF,” *arXiv preprint arXiv:2102.09980*, 2021.
- [6] G. Duan, Y. Fu, M. Cai, H. Chen, and J. Sun, “DongTing: A large-scale dataset for anomaly detection of the Linux kernel,” *Journal of Systems and Software*, vol. 203, p. 111745, 2023.
- [7] M. A. M. Vieira, M. S. Castanho, R. D. G. Pacífico, E. R. S. Santos, E. P. M. Câmara Júnior, and L. F. M. Vieira, “Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications,” *ACM Computing Surveys*, vol. 53, no. 1, pp. 16:1–16:36, 2020.
- [8] T. Høiland-Jørgensen et al., “The eXpress data path: Fast programmable packet processing in the Linux kernel,” in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, 2018.
- [9] J. Gallego-Madrid et al., “Machine learning-powered traffic processing in commodity hardware with eBPF,” *Computer Networks*, vol. 245, p. 110295, 2024.
- [10] J. Zhang, P. Chen, Z. He, H. Chen, and X. Li, “Real time intrusion detection and prevention with neural network in kernel using eBPF,” in *Proceedings of the 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2024, pp. 416–428.
- [11] T. Farasat, J. Kim, and J. Posegga, “SmartX Intelligent Sec: A security framework based on machine learning and eBPF/XDP,” *arXiv preprint arXiv:2410.20244*, 2024.
- [12] The Linux Foundation, “eBPF documentation and reference guide,” <https://ebpf.io>, 2024.
- [13] Aqua Security, “Tracee: Runtime security and threat detection using eBPF,” <https://github.com/aquasecurity/tracee>, 2024.
- [14] Sysdig, “Falco: Cloud native runtime security,” <https://falco.org>, 2024.
- [15] T. Hara and M. Sasabe, “Practicality of in-kernel/user space packet processing empowered by lightweight neural network and decision tree,” *Computer Networks*, vol. 240, p. 110188, 2024.
- [16] A. Brodzik et al., “Ransomware detection using machine learning in the Linux kernel,” *arXiv preprint arXiv:2409.06452*, 2024.