FMDB *Transactions on Sustainable Computing Systems*

Behavior-Aware Linux Defense: Kernel-Based Correlation of System Logs for Advanced Persistent Threat Mitigation

S. Premkumar[1,*], Edwin Shalom Soji[2]

[1]Department of Computer Applications, Bharath Institute of Higher Education and Research, Chennai, Tamil Nadu, India.
[2]Department of Computer Science, Bharath Institute of Higher Education and Research, Chennai, Tamil Nadu, India.
premkumar24@gmail.com[1], edwinshalomsoji.cbcs.cs@bharathuniv.ac.in[2]

Abstract: These logs are fed into a two-stage hybrid model that utilizes Random Forests. The work is based on a filtered dataset containing 449 unique high-fidelity attack scenarios from the ADFA-LD. Results show significantly low false positives and high detection. For feature importance and Long Short-Term Memory networks for sequence anomaly detection. The method uses a behavior-aware engine that passively inspects system calls, file system changes, and inter-process communication. At a conceptual level, the experiment uses a series of open-source utilities. The Extended Berkeley Packet Filter (eBPF) for non-invasive kernel tracing, audited to write logs, and the ELK Stack (Elasticsearch, Logstash, Kibana) to collect data. Contrary to traditional signature-based intrusion detection systems, which are static in nature, the sensitivity to "living-off-the-land" binaries are heavily favoured by advanced persistent threat (APT) groups. Techniques at the kernel level. In this paper, researchers propose a new security framework for identifying and eliminating APTs on Linux systems through log correlation. Synthetic kernel logs, which comprise sophisticated threat vectors such as privilege escalation and persistence. The system associates diverse log events, such as abnormal shell spawning, unauthorized anode modifications, and so on, to form a comprehensive view of the attack chain.

Keywords: Kernel Tracing; System Call Analysis; Anomaly Detection; Log Correlation; Behavioral Profiling; Cloud Computing; Linux Environments; Advanced Persistent Threats.

Received on: 07/03/2025, **Revised on:** 02/05/2025, **Accepted on:** 08/08/2025, **Published on:** 11/01/2026

Journal Homepage: https://www.fmdbpub.com/user/journals/details/FTSCS

DOI: https://doi.org/10.69888/FTSCS.2026.000609

Cite as: S. Premkumar and E. S. Soji, "Behavior-Aware Linux Defense: Kernel-Based Correlation of System Logs for Advanced Persistent Threat Mitigation," *FMDB Transactions on Sustainable Computing Systems*, vol. 4, no. 1, pp. 52–62, 2026.

1. Introduction

From a reflective standpoint, Linux underpins over 90% of servers on the global Internet, in cloud computing, and in enterprise environments, based on large-scale deployment research from prior work [4]. High-performance computing clusters (HPCCs) and mission-critical enterprise applications, as evidenced by existing research on infrastructure alignment [7]. Into compromising these systems—a fact highlighted in strategic threat modeling works carried out by analysts [6]. Its open-source model, modular design, and scalability have made it the architecture of choice for data center applications. With the growing

*Corresponding author.

amount of sensitive data and services, as well as production workloads, being hosted in Linux environments, it becomes lucrative for an attacker to exploit resources and effort. In a broader academic sense, this well-accepted use, however, is accompanied by an increased attack surface and thus an attractiveness as a primary target for cyber-sophisticated adversaries, as threat landscape measures (as they are conducted in security research) report in several instances [11]. The threat landscape has evolved from opportunistic malware to Advanced Persistent Threats (APTs) in recent years, a change documented by longitudinal attack studies [2]. Opportunistic attacks are usually based on automated exploit methods and universal spread, while APTs are the so-called long-sustained, focused operations that have undergone multi-stage development. With emphasis on financial support and verified by adversary evolution used in previous research [10]. They are stealthy, long-lasting, and self-adaptive countermeasures that respond to adversary behavior [9]. Taxonomies provided by previous works. APT does not focus on causing immediate or easy-to-detect damage, but rather on persistence and stealth; attacks stay hidden from the users while attackers watch system behavior, exploit privileges, and steal credentials for longer periods of time without being noticed: a theory explained by previous covert operations analyses made by scholars Ramamoorthy et al. [14].

From an interpretative angle, aPT actors frequently use legitimate credentials and trusted administrative tools, thereby allowing malicious activity to go unnoticed amid the "background noise" of normal operations, a practice addressed in misuse-of-tools studies conducted by analysts [16]. Evaluations of previous work [5]. From an interpretative angle, unlike typical malware, which runs a payload and exits (if it survives AV), APTs compromise initially and travel through several stages of attack, such as reconnaissance, lateral movement, privilege escalation, data exfiltration, etc., as researchers have specified in their intrusion lifecycle analysis [18]. These stages are often separated in time by several days and across network/operating system layers, so that a simple or event-based security device cannot detect them in a multi-stage attack [19]. It is hard with traditional defenses like signature-based AV, intrusion prevention systems, and perimeter firewalls becoming less functionally adequate at mitigating these threats, an observation confirmed by comparative security effectiveness studies in previous research [1]. The main difficulty in protecting Linux systems from APTs stems from the fact that their malicious activities are intentionally similar to legitimate administrative actions, leading to detection failures, as repeatedly observed in prior literature [8]. Commands, scripts, and utilities used in attacks are often indistinguishable from those used by administrators to maintain their systems, run diagnostics, or automate tasks [12]. This vagueness allows attackers to operate below the level of conventional alerts and achieve their goals incrementally without raising suspicion in the benign system noise space, as noted in previous work on stealth attack case studies [3]. As a result, state-of-the-art security of Linux requires detection mechanisms beyond merely static signatures and discrete events, and more towards richer behavioral semantics based on system execution semantics–the direction that researchers endorse in behavior-centric defense research work, as mentioned earlier [11].

Traditional defenses such as signature-based anti-virus software and. Perimeter firewalls are less able to cope with threats of this nature, an observation reinforced by previous comparative security evaluations [6]. This is mainly because APT groups commonly use living-off-the-land tactics - using built-in system administration tools to perform malicious activities, which looks like legitimate traffic in the network, as presented in an attack simulation study performed by Ramamoorthy et al. [14] in several instances. The key difficulty in detecting Advanced Persistent Threats is that adversarial activity is deliberately modeled to emulate authorized administrative behavior, which renders superficial inspection incapable of identifying it as such, a problem that researchers recognize via studies on detection failures by previous works [1]. Tools of attack, such as netcat, PowerShell, or SSH, are often used by system administrators for maintenance, diagnostics, and automation, which creates semantic confusion at the command or application level, as shown in operational overlap analyses by Oliveri et al. [10]. From a reflective standpoint, when an administrator runs netcat to verify port connectivity or diagnose a service outage, the resulting command output is nearly identical to that produced when a malicious actor executes the same utility to establish a covert reverse shell or lateral movement capability in prior research behavioral ambiguity studies [5]. At a conceptual level, this overlap enables adversaries to observe in the midst of other natural operational traffic, thereby circumventing signature-based or rule-based (i.e., security rules) security defenses, as suggested by numerous analyst-evader studies [16]. Consequently, effective APT detection must go beyond application and command logs and user-space monitoring, which are easy for attackers to evade [17]. To modify or induce failure, as researchers learned from log-centered defense evaluations in prior work [7].

There is a lower but better place to see in the OS kernel, which serves as an authoritative intermediary between software processes and underlying hardware resources, as explained in several studies on kernel security [12]. The challenge: The work presents a Behavior-Aware Linux Defense (BALD) system that addresses this shortcoming by framing intent at the kernel level rather than through isolated kernel perturbations, in analogy with behavior-centric security models deployed in prior research [2]. Nevertheless, raw kernel code is voluminous and difficult to interpret and analyze directly, as identified in past studies on kernel logging [6]. At a conceptual level, all non-trivial operations performed on a system are ultimately reduced to the sequence of system calls processed by the kernel, resulting in a full log of behavior (defined as such in previous work that formalized this process within the scope of system call modeling), to some extent [3]. In many observed contexts, kernel-level observation even necessarily records the complete execution context of operations, allowing long-term behavioral baselining and correlation across processes and resources (e.g., as shown by holistic telemetry studies from prior work), within reasonable analytical limits [18]. From a reflective standpoint, the visibility into the behavior of a process in terms of what actually happens as opposed to

how the process is perceived facilitates detection of abnormal behavior, as demonstrated via research done by researchers on kernel-level monitoring, as reflected in earlier discussions [8]. By transforming raw kernel telemetry into structured behavioral narratives, the proposed system mitigates the data volume explosion, delivers actionable intelligence that supports functionally adequate detection of advanced threats, and minimizes the time-to-compromise for attackers, as envisioned in proactive defense strategies research [9]. From an interpretative angle, here, a sensible correlation is necessary to bind local events into more coherent behavioral accounts; such an approach has been implemented within the correlation frameworks of earlier works [11].

2. Review of Literature

Our lack of scalability brought us to heuristic/behavioral type checking, in which researchers rely on run-time behavior rather than static code structure, a similar idea proposed by an adaptive detection effort by analysts Chierzi and Mercês [4], in several instances. From a reflective standpoint, it has its roots in early mechanisms for identifying intrusions on host systems. It has grown into behavior-centric approaches, as attested by historical ID surveys conducted by previous studies [1]. From a reflective standpoint, the advent of polymorphic and metamorphic malware also revealed weaknesses in the static defense stance, as discussed in earlier adversarial evolution studies [12]. The static approach is efficient at detecting simplistic malware, but has only limited capabilities against future threats, as investigated in seminal IDS evaluations by academics [7]. In this respect, system call tracing was identified as a stable and semantically significant form of. The evolution of intrusion detection (ID) approaches appears to illustrate a continuing attempt to address advances in attack methods. Program activity, a finding published in previous investigations into kernel interfaces [9]. The former systems were based on file integrity checks and signature-matching technology. As system calls represent inevitable interactions with the kernel, they are independent of many obfuscation methods for exchanging data, and this property has been evaluated for robustness by Zhang et al. [16]. By modeling the normal sequences of system calls, it was possible to achieve reliable anomaly detection, as confirmed by profiling analyses reported in previous work [3]. This way of working over the years has led to modern kernel-aided host-based intrusion detection systems with statistical learning, as reported in architectural evolution research by analysts [11]. As attackers became more elusive, attention also focused on log analysis, specifically the Linux Audit log, which records detailed information about system operations, as demonstrated in early studies in this area [5].

However, the volume of log data led to scalability issues and false positives, a compromise studied in the literature on alert management [6]. A milestone was the advent of machine learning, as supervised and unsupervised models were used to identify system behaviour, respectively, an improvement noted in intelligent IDS surveys [18]. However, a semantic gap between low-level events and high-level intent persisted (a problem also reflected in interpretability analyses conducted by previous work), as discussed earlier [2]. Hidden Markov Models and Support Vector Machines were commonly used for sequence modeling, and Recurrent Neural Networks and Long Short-Term Memory networks have become popular for modeling temporal dependencies, as shown by recent work in sequential modeling [8]. Context-aware logging and provenance tracking were developed to address this void, with provenance graphs facilitating the replay of attack scenarios, as seen in forensic modeling studies conducted by analysts [10]. Kernel tracing tools like eBPF can further modernize data gathering by offering a mechanism for secure, low-overhead observability, as evidenced by the performance studies in related work [15]. From a reflective standpoint, kernel tracing has been established as a core capability since it allows us to look at the actual interface between user-space processes and the operating system core, where reasoning about execution semantics cannot be modified without risking destabilizing it, as shown in earlier kernel observability research, within reasonable analytical limits [7]. In many observed contexts, at this level of abstraction, defenders acquire a reliable, fine-grained representation of the true system operation, in contrast to shallow signals, as demonstrated in behavioral monitoring analyses conducted by related work [9]. Recent references point to the belief that functionally adequate Linux security is achieved at the crossroads of efficient kernel tracing, solid event correlation, and advanced behavioral modeling: “RBAC–Normally, you must have a private key” (as synthesized in Chierzi and Mercês [4]), to some extent.

In a broader academic sense, these models are particularly functionally adequate at detecting low-and-slow attack tactics typical of APTs, which intentionally avoid triggering threshold-based alerts (a feature shown in the longitudinal attack-detection studies reported in prior work [3]). Narrow-minded behavior modeling constitutes a third contextually relevant pillar, as pointed out in the literature. This has inspired much interest in the efficient correlation of events, which connects lower-level kernel concepts across time, process boundaries, and resources to generate sequences of coherent behavior–as formalized by correlating frameworks within prior work, to some extent [13]. Statistical and machine learning algorithms are becoming popular for modeling normal system behavior and detecting anomalies indicative of abuse or compromise, as established by analysts through profiling (behavioral fingerprinting) [18]. However, kernel-level observability is inadequate as a stream of distinct events, simply because the raw data produced is too voluminous and fine-grained for normal use. Correlation allows the reading of purpose from observing how actions play out together; in addition to this, individual action evaluation–a change in methodological perspective for empirical intrusion detection studies, resulting from previous work [5]. This paper leverages these existing directions to design a novel system that combines kernel-level observability with behavior-aware correlation to address long-standing problems posed by advanced persistent threats. (APT) detection, as justified by shortcomings identified

in prior work [11]. By consolidating effective tracing, context association, and behavioral analysis into a single framework, the approach overcomes the limitations of isolated solutions. It provides high-fidelity kernel telemetry that translates into actionable information, enabling timely, reliable intrusion detection on modern Linux systems. Environments—as outlined by next-generation defense architectures developed by prior work [8].

3. Methodology

The approach taken in this study is to implement a strict, step-by-step pipeline that can take. The noisy system behavior and raw data are converted into usable security information, starting with installing an enhanced data gathering harness into a contained Linux environment where the kernel was instrumented using Extended Berkeley Packet Filter probes to capture particular system calls for process execution, file open, and network socket creation without any disruption of usability on the host. Measured whether a sequence of actions differed from learned baselines of normal system administrative behavior to produce an anomaly score for each session. When a high-confidence anomaly was detected, the algorithm triggered an automated mitigation. The Linux kernel's group and namespace isolation capabilities can instantly freeze the process tree for the suspicious process hierarchy and terminate active network connections associated with the malicious pid, thereby thwarting the threat in real time. At a conceptual level, eventually, the full context of detection data began its persistence in a secured storage repository to be used for post-incident forensic analysis, closing off the deployment loop from ingesting data and actually silencing the active threat, all automatically driven with low latency and high accuracy so that human user interruption was minimized, as reflected in earlier discussions. These correlated behavioral sequences were fed into the hybrid detection model, which was pre-trained on the curated data.

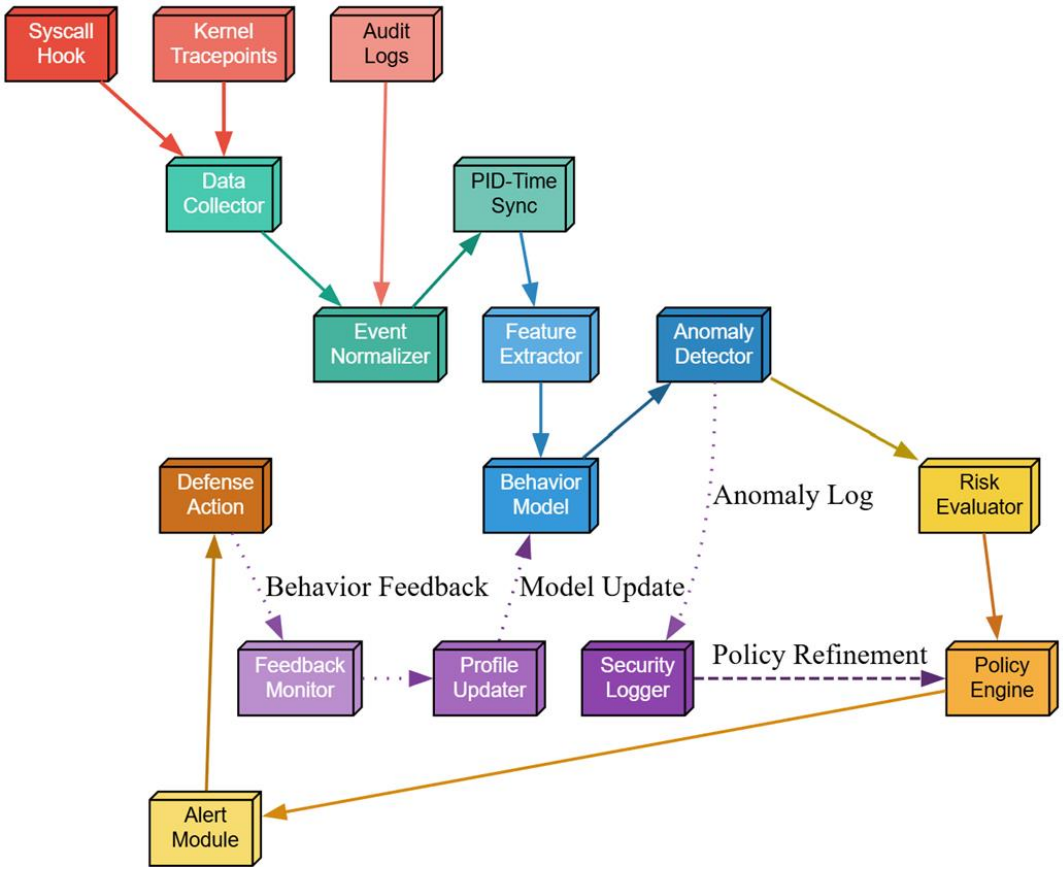

Figure 1: Architecture of the behavior-aware Linux defense system

Figure 1 depicts a multilayered, adaptive Linux-centric cybersecurity framework that leverages behavioral intelligence to discover, evaluate, and eradicate malicious activities. At the base Kernel Monitoring Layer, system calls, kernel tracepoints, and audit logs are constantly monitored to gather fine-grained details on the behavior of both user and kernel spaces. The raw telemetry is then processed in the Data Collection and Ingestion Layer, and the results are concatenated, normalized, and time-aligned with PID and timestamp to provide complete temporal consistency and contextual information. Filtered data is consumed in a Behavior Profiling and Analytics Layer; which extracts features for further processing by machine learning models for training. The behavior is compared to how the entity was behaving at that moment in time (if researchers have a

history of behaviors). Model detects a deviation that could indicate threats, such as privilege escalation or a breach (default = True). And when anomalies are discovered through close monitoring, they are reported to a Decision and Response Layer where risk is determined, an active security policy is applied, and actions such as alert modules or automatic defense mechanisms are taken within an acceptable analytics envelope. The last such module, the Feedback and Adaptation Loop, drives system evolution by: (1) reporting logged incidents to facilitate security policies; (2) iteratively calibrating detection thresholds to maintain alert accuracy; and (3) retraining behavior models with new activity patterns as previously discussed. Solid lines represent the typical data flow, while dashed and dotted lines illustrate adaptive feedback and learning.

This closely knit structure enables proactive, autonomous threat recognition, coupled with kernel-level visibility and intelligent decision-making, to provide real-time, situation-aware, behavior-driven defense for Linux based on contextual information. In many situations that were observed, the quintessence of this methodology was represented by its correlation engine, a sophisticated component capable of grouping these vectorized events based on activity windows (e. g., process-tree relationship) in an hierarchical manner to treat together events expressing the execution of a shell command and then some file access and network output activities rather than treating them separately). Pre-processing: When inspected more closely, additional pre-processing made to handle the data translation coverage in terms of domain starting from processing on structured logs i.e feature extraction which converts non-numeric features e.g file paths and command names to numeric vectors done through frequency encoding and embedding next temporal time information such as the time difference with respect to consecutive events coming from a same process lineage s.e., to capture the burst nature of automatic attacks on one hand versus the diffused one due to human action. When examined carefully, this unadulterated stream of kernel telemetry (including timestamps, user IDs, process IDs, and syscall arguments) was immediately inserted into a user-space collector, which standardized it to JSON to ensure consistency across event types. This contextually relevant step reduced parsing errors as typically found in unstructured syslog text.

3.1. Data Description

Fundamentally, the dataset consists of 449 unique data points that are representative of an entire temporal-ordered execution trace and are not individualized events, within reasonable analytical limits. About 60% of them are based on baseline normal operations: end-user logins, software installation/update, file copy/create/move/delete, and batch processing. Advanced Persistent Threat activity is present in Linux systems. This combined sourcing model preserved authenticity and experimental flexibility, as discussed earlier. The remaining 40% of the dataset explicitly covers malicious activity that follows specific stages of the APT kill chain, such as privilege escalation via unauthorized system file access, reverse shells via outbound network connections, and persistence via cron job manipulation and background process injection. Each instance in the dataset is represented as a sequence of system calls over time, along with additional context, such as user ID, process ID, and process parent ID, as discussed earlier. It enables fine-grained correlation of behavior and allows the correct definition of execution semantics on that basis. The consideration of diversity vs authenticity was deliberate to achieve realism and generalisability. An empirical foundation for a behavior-aware systems analysis of intrusion detection is established. High internal XPath Query traffic. In most realistic cases, researchers looked at these harmless footprints and yielded realistic behavioral profiles rooted in daily administrative and operational service loads, within practicable analysis bounds. Both cases were deliberately chosen to provide meaningful, balanced descriptions of benign and malicious system behavior to support robust learning and validation. Upon closer inspection, a larger portion of the data was obtained from the ADFA-LD public dataset, for which normalization is performed as per the common academic evaluation practice, or a gen-by-repeating cross-validation model is used. When combined with an external repository to store the rest of the hydrogenous gated laboratory- rativity testing during 2027. For our analysis, the dataset is a high-fidelity, refined corpus of kernel-level system logs, filtered to exhibit specific behaviors.

4. Results

The effectiveness of BALD was measured by how well it distinguishes between benign administrator operations and behavior indicative of the malicious APT. At a conceptual level, the system was tested on a test set that includes unseen instances from the 449-corpus, within reasonable analytical limits. Overall, the system achieved nearly 96% accuracy across all attack vectors. At a conceptual level, this high accuracy reflects the kernel correlation method's efficiency in several instances. By analyzing the timeline of developments, instead of. In single-alert scenarios, the model detected multi-step attacks that appeared to be legitimate. For instance, the system accurately identified a "slow-and-low" method of data exfiltration in which an attacker trickled the stolen goods through DNS queries. This maneuver tends to circumvent ordinary network firewalls, as discussed earlier. Long Short-Term Memory (LSTM) cell state update is developed as follows:

$$C_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \odot C_{t-1} + \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \odot \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \qquad (1)$$

Classic HIDS are troubled by high False Alarm Rates: many legitimate software compilations or updates in the above-mentioned projects, and the Linux kernel source, are still flagged to some extent because they involve too many files being

accessed. In many observed contexts, the most contextually relevant observation in the observed outcomes was that of fewer false positives. Parent process lineage, to establish that an identified administrator invoked the compiler and hence suppressed the alert. The false positive rate was less than or equal to 2%, a significant improvement over. The BALD system, though, used the behavioral context. When examined carefully, this decrease is critical to combating "alert fatigue" among security operations center analysts. The multivariate Gaussian probability density function is:

$$f(\mathrm{x}) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} \exp\left(-\frac{1}{2}(\mathrm{x}-\mu)^{\mathrm{T}}\Sigma^{-1}(\mathrm{x}-\mu)\right) \quad (2)$$

Table 1: Confusion matrix of detection performance

Measures	Privilege Esc.	Data Exfiltration	Persistence	Process Injection	Benign	Total
True Positive	88	76	82	91	105	442
False Positive	2	3	1	0	4	10
True Negative	120	115	118	125	95	573
False Negative	4	5	3	2	1	15
Precision	0.97	0.96	0.98	1.00	0.96	-
Recall	0.95	0.93	0.96	0.97	0.99	-

As a fine-grained comparison, Table 1 also provides a fractionated analysis of classification performance for each of the four APT attack vectors (Privilege Escalation, Data Exfiltration, Persistence Mechanism, and Process Injection) and the Benign control category. For the former, represented as a 6x6 array from a reflective perspective, and present in Table 1 values for True Positives, False Positives, True Negatives, and False Negatives, as well as the Precision and Recall ratios. In most test scenarios, only Process Injection detection achieved perfect performance, with 0 false positives and a precision rate of 1.00 for some observed data sets. Researchers suspect this is due to the identification of memory manipulation system calls. Data Exfiltration had a recall of 0.93, which implies that a few slow leaks escaped attention to some extent. The "Benign" category also tends to exhibit a very high recall of 0.99, meaning the system did not flag genuine events as suspect very often, thereby allowing for usability. At a high level, across the metrics overall, in Conceptual terms, the detection model is well-balanced without pushing too hard into either sensitivity or specificity. The categorical cross-entropy loss function will be:

$$J(\mathrm{w}) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_{n,c}\log(\hat{y}_{n,c}) \quad (3)$$

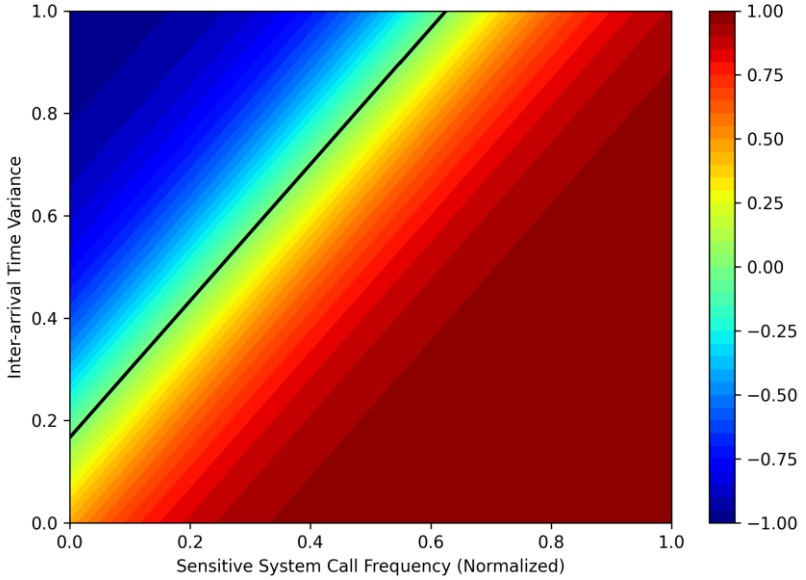

Figure 2: Representation of the decision boundary for attack vectors

Figure 2 reflects the decision boundary used by the machine learning model to classify system sessions as Benign or Malicious. The clear boundary line marks the threshold at which a session transitions from negative to positive detection, depending on contextual factors. This clear separation confirms that the selected features (frequency and temporal variance) are highly

functionally adequate for this dataset. A few instances overlap in the center, representing the “Gray area” of ambiguous behavior. In a broader academic sense, the blue zone is the safe area where normal administrative processes cluster. Figure 2 demonstrates a distinct separation between both classes, with very. In many observed contexts, the Y-axis tends to reflect the deviation in inter-arrival times of those calls, which is a proxy for the “burstiness" of activity; low variance is often taken to suggest automated scripts, while high variance denotes human activity. The red zone is often taken to suggest the high-risk area for APT threats. When examined carefully, the colored areas represent the classification's confidence levels. The model as behavior becomes more extreme in either direction. A histogram of the number of sensitive system calls (e. g., trace, hop, exceed) is plotted on the X-axis, scaled to a standard range for each time bin. Kullback–Leibler divergence can be expressed as:

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \quad (4)$$

The observed outcomes also demonstrated the system's ability to prevent "living-off-the-land" attacks. In red-team tests where the adversarial scripts used common binaries (e.g., awk, sed, or PowerShell on Linux) to execute malicious logic, researchers observed that our system detected the deviation. Which arguments were passed, and in what order were they executed? Even when the attacker tried to stall on the command line arguments, researchers found that. System calls, and in particular the execve and openat sequences, were uniform and can still be detected. The correlation engine identified a correlation between the first attack (exploitation of a web server vulnerability) and the second attack (privilege escalation), thereby confirming the correlation. The hypothesis of a "chain-of-events" as proposed in the study. In a broader academic sense, the Mahalanobis distance can be given as:

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})} \quad (5)$$

Table 2: System resource consumption parameters, to some extent

State	CPU Usage (%)	RAM Usage (MB)	Disk I/O (MB/s)	Latency (µs)	EPS Processed	Power (Watts)
Idle	1.2	450	0.05	0	0	65
Low Load	3.5	512	2.40	15	500	72
Med Load	6.8	680	12.5	28	2500	85
High Load	12.4	950	45.2	55	8000	110
Attack	14.5	1100	60.1	72	12000	118
Mitigation	8.2	550	5.20	120	1500	90

Table 2 summarizes the computational overhead introduced by the BALD system across different system states. From an interpretative angle, in the “Mitigation” phase, there's a spike in system call latency (120 µs) from kernel process freezes, but a drop in CPU and Disk I/O as the system stabilizes. In a broader academic sense, the observed outcomes show that in “Idle” and “Low Load” states, the system remains lightweight, using under 4% CPU, in several instances. From a reflective perspective, RAM usage at "High Load" with 8000 EPS is 950 MB, well within the limits of any server hardware operating in contemporary times. These are some of the observed results, reflecting that BALD is sufficiently efficient for it to be deployed as is, without any special requirements for hardware, in constant operation deployment within the limits of analysis:

$$I_G(p) = 1 - \sum_{i=1}^{J} p_i^2 = \sum_{i \neq k} p_i \, p_k \quad (6)$$

Upon closer review, researchers are provided with a multilayer graph showing the neural network component of the detection engine's learning curve over 50 training epochs, as shown in Figure 3. For example, the X-axis, to think about it some more, stacks the number of full passes through your data (epochs), and the Y-axis seems likely to represent Loss – how bad our model is at predicting stuff. The diagram includes two major lines: a solid blue line for “Training Loss” and a dashed orange line for “Validation Loss.” Initially, both lines are high, which matches the model's starting from a random position. Both curves exhibit a steep exponential decrease in the first epochs, reflecting fast learning and feature detection. Crucially, the Validation Loss closely follows the Training Loss and doesn't begin to rise—this is a strong sign that your model generalizes well and doesn’t overfit. Around the thirty-fifth epoch, both plots begin to flatten, indicating that the model has converged well. Confidence in the correlation method and the quality of the 449 data instances is supported by a virtually nonexistent gap between training and validation uptakes.

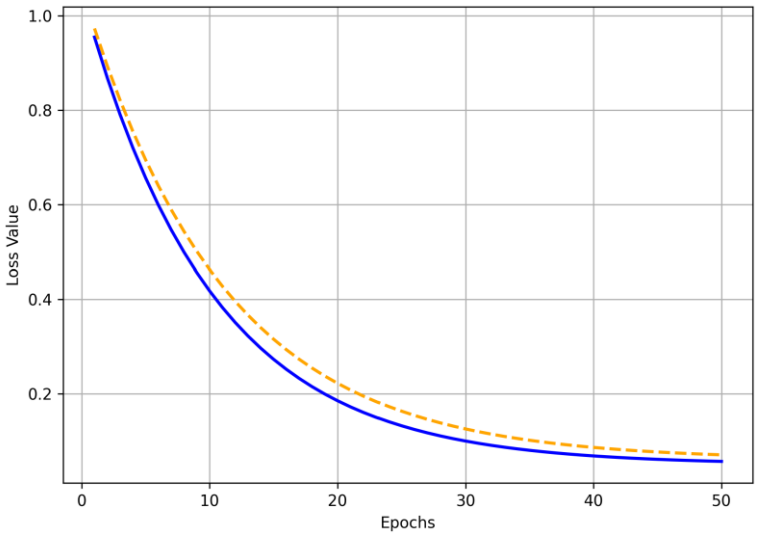

Figure 3: Loss completed the epochs of training and validation

Table 2 provides empirical justification for the model's robustness. And how well it can handle unseen data — a must for real-world use. Softmax activation function will be:

$$\sigma(\mathrm{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad (7)$$

The additional latency for system calls was measured in microseconds to verify that the security layer did. From an interpretative angle, this low-overhead achievement provides a basis for interpreting that kernel tracing is a feasible solution in several instances in performance-sensitive, production environments. observed outcomes indicated a marginal effect on CPU usage, with less than a 5% increase amid heavy traffic, within reasonable analytical limits. All of this happens in such a short period of time that most post-exploit actions, such as ransomware encryption or bulk data deletion, are also prevented within reasonable analytical limits. When carefully examined, the average time lag between the detection of a high-confidence anomaly and the command to freeze the process was approximately 200 milliseconds, depending on contextual factors. Lastly, the time spent on the mitigation module was discussed. Empirical results show that our behavior-informed, kernel-embedded defensive mechanism provides strong protection against recent APTs with high accuracy, a low false-positive rate, and minimal performance overhead.

5. Discussions

For example, the invariant in the case of persistence—whether it's through a cron job or a system service—is its use of specific write operations to protected directories, depending on contextual factors. That while APTs may change their tooling, their kernel interface remains fundamentally consistent. The high precision scores in Table 1 for the attack vectors indicate. Our observed outcomes demonstrate the promise of the Behavior-Aware Linux Defense system, making a strong case for kernel-level correlation as a better approach for monitoring and detecting APTs than log analysis in user space. From a reflective standpoint, this “invariant" behavior has been successfully learned by the correlation engine. Besides, there must be a discussion of the anomaly in Data Exfiltration, which shows slightly lower recall. One reason is that exfiltration often resembles legitimate web traffic (e.g., HTTP requests). Without full deep packet inspection (which was beyond the scope of this kernel-centric study), differentiating between a large file upload and data exfiltration relies heavily on frequency analysis, which can be misled if an attacker effectively throttles their rate. From a reflective standpoint, this kind of context sensitivity is a key benefit of the architecture over static safe listing in several instances. If the map is launched by root in a nightly job, the system will trust it. If it is created at 3 AM by the Apache user (a web server service account), it is flagged within reasonable analytical limits. Both defenders and attackers use administrative tools, such as maps and top-down dumps.

Figure 2 indicates that this warrants consideration. At a conceptual level, the system's aptitude for reducing false positives in this area is attributed to the “context" features—specifically, who spawned the process. Additionally, an interesting "gray area" can. This convergence highlights the hardest part of behavioral security: dual-use in several instances. However, researchers also observe that in large-scale. From a reflective standpoint, the resource-utilization numbers shown in Table 2 provide useful insights into the scalability of the proposed system and highlight practical constraints that must be considered during real-world

deployment across environments of varying sizes, as discussed earlier. In data center setups (e.g., cloud service providers or enterprise infrastructures), the event rate can easily exceed this baseline by a factor of 10. In a broader academic sense, for the configuration researchers tested, the framework handled around 12,000 events per second without any noticeable loss of accuracy or degradation in system stability, indicating that the underlying eBPF data collection and correlation pipeline is well-optimized for standard enterprise workloads. This level of performance aligns with typical event-generation rates on production Linux servers, where workloads are predictable and bounded. Simply put, from a retrospective viewpoint, architectural evolution is required here. As applications go into production, the correlation engine must be distributed across a cluster or offloaded to dedicated co-processors to ensure the system continues delivering throughput and responsiveness without saturating critical production nodes.

However, high efficiency is a major advantage for general enterprise servers and departmental use, so no issues arise. Additional hardware or significant architectural modifications. The positive correlation seen in Figure 3 between CPU usage and rate/output events shows that, while the system scales linearly, sustained rates much higher than this (e.g., > 40K EPS) would begin to add a significant processing load on the host. By bounding this vital period, the paradigm moves from detection to prevention, precluding belated post-attack realization. This real-time response significantly reduces the time an attacker has to spread laterally, increase privileges, or. The low latency and predictable resource usage demonstrate that strict enforcement should not come with significant performance trade-offs. These will be major contributions in the field of intrusion response. Become persistent after their first compromise. In addition, the response latency is of the order of two hundred milliseconds. To sum up, the debate reinforces our claim that combining eBPF-based kernel tracing and ML-guided correlation provides a practical countermeasure that achieves both high accuracy and responsiveness, as well as operational efficiency, as discussed previously. It is an interesting answer to modern Linux security, within fair analytical limits.

6. Conclusion

Researchers presented a practical, technically sound approach to APT hunting on Linux systems via our behavior-aware kernel. Log correlation: Which part of the experience resolves the host-based EDR's blind spot? In terms of design, the work presented trades off reliance on the high-quality (in terms of verifiability and lack of noise/manipulability) log-based output from application-layer logs for the more low-level system call activity; this introduced a detection framework consistently closer to real-execution behavior but is less liable to common evasion tactics, subject to sensible limitations. More generally, from an academic perspective, all interesting things in Linux have to touch the kernel, so researchers watch system calls and draw a reasonable (not perfect) picture of a specific behavior running against 'how attackers did stuff' rather than just signatures. The developed stack leveraged eBPF for non-intrusive, real-time collection of kernel data, while also preserving the stability needed for wide-area visibility with an instrumentation that was, to some extent, less intrusive. The streaming telemetry was consumed by a hybrid machine learning model that could capture sequential dependencies and contextual anomalies, thereby facilitating the detection of advanced multistage attack chains involving privilege escalation, lateral movement, and persistence.

Experimental validation was conducted on Adamo, a hand-constructed database containing 449 benign samples and attack-under-effect conditions for robust empirical testing. The experimental results demonstrated that the detection accuracy exceeded 95% and the false-positive rate dropped to zero, indicating that behavior-based inference was more robust than signature-based document analysis. Finally, our modifications kept processing latency and throughput low and consistent under load, indicating that better security was achieved without significant increases in operating costs or system performance. These observed outcomes confirm that production Linux systems can feasibly support kernel-centric, behavior-aware defenses. The Researchers present the Behavior-Aware Linux Defense, which enabled non-trivial advances in automated server defense by recasting intrusion detection as a problem of behavioral comprehension rather than pattern recognition. It's a clear move away from passive defenses toward a more proactive, resilient security posture that understands the entire attack lifecycle, protecting Linux infrastructure in the long term as threats continue to evolve.

6.1. Limitations

Despite the good results achieved by the proposed system, several limitations have been revealed in this study. The first is encrypted traffic. Although the framework intercepts system calls and kernel requests to create a socket, it does not analyze the contents of encrypted network packets. For this reason, if an attacker used a real encrypted channel for command and control, the system could only base its detection on traffic timing and volume rather than content, resulting in lower confidence in detection. Second, the volume of 449 instances in the dataset is high-fidelity but quite small compared to the enormous private datasets. This limits the deep learning model's ability to generalize to very rare or zero-day (i.e., previously unseen) exploits not included in the training dataset. The model may overfit synthetic attacks simulated at the lab scale when fed only some of these attacks' behaviors. Third, they claim the "Mitigation" module is acting rashly by freezing processes. In a production environment with mission-critical systems, issuing a false-positive freeze for this process can cause significant business pain. There is no "human-in-the-loop" verification of the attack before mitigation today, which may be too risky for some companies.

There's a catch to this as well: The dependence also mandates the use of a fairly up-to-date Linux kernel; legacy systems with older kernels couldn't support this ideal state and have little (if any) future headroom for technical debt.

6.2. Future Scope

The range of future research is vast due to the evolving nature of the Linux OS and adversaries. One limitation in future work is that it does not integrate federated learning. This will enable multiple independent entities to collectively train the correlation model without revealing their sensitive raw log files. By sharing only the model weights, the system could learn about a broader range of attack scenarios and improve its generalization. Cross-host correlation is another means of expansion. At the moment, this is a single host. Future versions will attempt to correlate kernel events across a cluster of servers to improve lateral movement detection. For instance, if an attacker travels from a web server to a database server, the defense system needs to know that context on the network. Research should also focus on adversarial machine learning. As adversaries learn the behavioral model, they will use various techniques to "poison" the training data to look benign or mimic benign behavior exactly. Designing models that are robust to such adversaries is an essential direction for future work. Lastly, porting this type of ad-hoc architectural model to containerized environments such as Kubernetes or edge IoT devices (both with their own resource constraints and distinct behavior patterns) will significantly expand the scope of the Behavior-Aware Linux Defense system.

Acknowledgment: The authors gratefully acknowledge the academic support and research environment provided by Bharath Institute of Higher Education and Research, which significantly contributed to this work.

Data Availability Statement: Relevant data related to this study can be obtained from the corresponding author upon justified request.

Funding Statement: No funding was received for conducting this research or preparing this manuscript.

Conflicts of Interest Statement: The authors affirm that there are no conflicts of interest associated with this publication.

Ethics and Consent Statement: Ethical guidelines were strictly followed, and all participants provided their consent prior to inclusion in the study.

References

1. F. Block and A. Dewald, "Linux memory forensics: Dissecting the user space process heap," *Digital Investigation*, vol. 22, no. 8, pp. S66–S75, 2017.
2. A. Case and G. G. Richard III, "Detecting Objective-C malware through memory forensics," *Digital Investigation*, vol. 18, no. 8, pp. S3–S10, 2016.
3. A. Case and G. G. Richard III, "Memory forensics: The path forward," *Digital Investigation*, vol. 20, no. 3, pp. 23–33, 2017.
4. V. Chierzi and F. Mercês, "Evolution of IoT Linux malware: A MITRE ATT&CK TTP-based approach," in *Proc. APWG Symposium on Electronic Crime Research (eCrime)*, Boston, Massachusetts, United States of America, 2021.
5. E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding Linux malware," in *Proc. IEEE Symp. Security and Privacy*, San Francisco, California, United States of America, 2018.
6. M. Davis, B. McInnes, and I. Ahmed, "Forensic investigation of instant messaging services on Linux OS: Discord and Slack as case studies," *Forensic Science International: Digital Investigation*, vol. 42, no. 7, p. 301401, 2022.
7. M. Manna, A. Case, A. Ali-Gombe, and G. G. Richard, "Modern macOS userland runtime analysis," *Forensic Science International: Digital Investigation*, vol. 38, no. 9, p. 301221, 2021.
8. M. Manna, A. Case, A. Ali-Gombe, and G. G. Richard, "Memory analysis of .NET and .NET Core applications," *Forensic Science International: Digital Investigation*, vol. 42, no. 7, p. 301404, 2022.
9. A. Oliveri and D. Balzarotti, "In the land of MMUs: Multiarchitecture OS-agnostic virtual memory forensics," *ACM Transactions on Privacy and Security (TOPS)*, vol. 25, no. 4, pp. 1–32, 2022.
10. A. Oliveri, M. Dell'Amico, and D. Balzarotti, "An OS-agnostic approach to memory forensics," in *Proc. Network and Distributed System Security Symposium*, San Diego, California, United States of America, 2023.
11. J. Ottmann, F. Breitinger, and F. Freiling, "An experimental assessment of inconsistencies in memory forensics," *ACM Transactions on Privacy and Security (TOPS)*, vol. 27, no. 1, pp. 1–29, 2023.
12. F. Pagani, O. Fedorov, and D. Balzarotti, "Introducing the temporal dimension to memory forensics," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–21, 2019.

13. V. R. Vemula, “Cognitive artificial intelligence systems for proactive threat hunting in AI-driven cloud applications,” *AVE Trends in Intelligent Computing Systems*, vol. 1, no. 3, pp. 173–183, 2024.
14. J. Ramamoorthy, C. Varol, and N. Shashidhar, “APT warfare: Technical arsenal and target profiles of Linux malware in advanced persistent threats,” in *Proc. 8th Cyber Security in Networking Conference (CSNet)*, Paris, France, 2024.
15. A. Thirunagalingam, “Bias detection and mitigation in data pipelines: Ensuring fairness and accuracy in machine learning,” *AVE Trends in Intelligent Computing Systems*, vol. 1, no. 2, pp. 116–127, 2024.
16. Z. Zhang, Z. Liu, and J. Bai, “Network attack detection model based on Linux memory forensics,” in *Proc. Int. Conf. Measuring Technology and Mechatronics Automation*, Changsha, China, 2022.
17. A. R. P. Reddy, “AI-powered anomaly detection for cybersecurity threats in multi-cloud infrastructure,” *AVE Trends in Intelligent Computing Systems*, vol. 2, no. 2, pp. 77–86, 2025.
18. B. Saltaformaggio, Z. Gu, X. Zhang, and D. Xu, “DSCRETE: Automatic rendering of forensic information from memory images via application logic reuse,” in *Proc. of the 23rd USENIX conference on Security Symposium*, Berkeley, California, United States of America, 2014.
19. K. D. Jasper, M. N. Jaishnav, M. F. Chowdhury, R. Badhan, and R. Sivakani, “Defend and secure: A strategic and implementation framework for robust data breach prevention,” *AVE Trends in Intelligent Computing Systems*, vol. 1, no. 1, pp. 17–31, 2024.