Post-Quantum Cryptographic Approach for Linux Kernel Module Used in Intrusion Detection System

Mariuca PRICOP, Cristian TOMA
Bucharest University of Economic Studies, Romania
pricopmariuca21@stud.ase.ro, cristian.toma@ie.ase.ro

As quantum computers get closer to being used in real-life situations, existing methods to protect against cyber threats are becoming less effective. This paper presents a hybrid, quantum-enhanced intrusion detection system implemented as a Linux kernel module, integrating kernel-level monitoring, user-space intelligence, quantum machine learning (QML) and post-quantum cryptographic (PQC) signature verification. The system uses a zero-trust approach by requiring cryptographic authorization for legitimate actions while simultaneously detecting anomalous behavior. Final enforcement decisions are executed within the kernel, ensuring low-latency response and strong security guarantees. By decoupling heavy cryptographic operations from kernel space and leveraging quantum-enhanced analysis techniques, this approach achieves a balance between performance, scalability, and resilience against future quantum threats.

Keywords: *Intrusion Detection System, Linux Kernel Module, Post-Quantum Crypto*
DOI: 10.24818/issn14531305/30.2.2026.06

1 Introduction

Intrusion Detection Systems (IDS) are an essential part of modern cybersecurity. They continuously monitor both system and network activity to spot signs of malicious behavior, policy violations, or attempts to access resources without permission. Depending on what they observe, IDS solutions are typically categorized as either host-based (HIDS), which focus on activity within a single system, or network-based (NIDS), which analyze traffic moving across a network. To detect threats, these systems usually rely on two main approaches: signature-based detection, which looks for known attack patterns, and anomaly-based detection, which identifies unusual behavior that deviates from what is normally expected [1]. However, traditional IDS methods often have trouble identifying new or previously unseen attacks, and they can produce a high number of false positives. This reduces their overall effectiveness, especially in fast-changing or highly dynamic environments.

To improve visibility and detection accuracy, recent research has focused on integrating IDS mechanisms deeper into the operating system, specifically at the kernel level. Monitoring at this level provides direct access to system calls, process behavior, and network packet flows, enabling more precise and tamper-resistant detection than user-space approaches. By capturing events such as process execution or network communication right at the source, kernel-based IDS solutions can make and enforce security decisions in real time with minimal delay [2].

Related research has explored several complementary strategies for enhancing kernel-level security. Early syscall-mediation systems such as Systrace (2003) introduced systematic confinement and policy enforcement at the system call interface, enabling runtime decision-making at the kernel boundary. More recent Linux mechanisms like seccomp user notifications similarly allow selected system calls to be delegated to a supervising user-space process, though they explicitly warn against using this mechanism as a comprehensive security framework [3]. In parallel, modern runtime-security solutions such as Falco and Tetragon leverage kernel drivers or eBPF hooks to capture detailed kernel-context telemetry and perform behavioral analysis in user space, with optional support for in-kernel enforcement actions.

A complementary line of work focuses on cryptographically authenticated actions,

where security-relevant operations must be validated before the kernel allows them to proceed. A notable example is Authenticated System Calls (2005), which extends each system call with a policy descriptor and a message authentication code that the kernel verifies prior to execution. Likewise, Linux integrity mechanisms such as IMA appraisal and kernel module signing embody the principle of kernel-resident verification, ensuring that untrusted user-space components cannot influence the authenticity of sensitive artifacts [4]. Recent upstream developments also integrate post-quantum signature systems such as ML-DSA (standardized by NIST in FIPS 204) into module authentication, reflecting a broader move toward quantum-resistant in-kernel validation mechanisms.

At the same time, the rise of quantum computing introduces new challenges for cybersecurity. Many of today's widely used cryptographic algorithms are expected to become vulnerable to quantum attacks [12], which has driven the development of post-quantum cryptography (PQC) as a more resilient, future-proof alternative. In parallel, quantum machine learning (QML) is being explored for its potential to strengthen anomaly detection in security systems, offering new ways to identify threats that traditional methods may miss.

Despite these advancements, existing intrusion detection systems typically treat anomaly detection and action authentication as separate concerns, lacking a unified framework that combines behavioral analysis with strong cryptographic guarantees. Additionally, many solutions only rely on classical cryptographic primitives that may become vulnerable in the near future.

To address these limitations, this paper introduces a **hybrid intrusion detection and firewall system** implemented as a Linux kernel module. The proposed solution combines low-level kernel monitoring with quantum-inspired anomaly detection and post-quantum cryptographic signature verification. By verifying ML-DSA signatures directly within the kernel and coupling this with user-space behavioral analysis, the system creates a layered, zero-trust architecture that both identifies anomalous behavior and enforces strict cryptographic authorization for sensitive actions. Through this fusion of kernel-space visibility, quantum-resistant authentication, and advanced detection techniques, the design aims to deliver a scalable and future-proof security solution for operating system protection.

2 Linux Kernel Components Architecture

Modern operating systems are built on a clear separation between user space and kernel space, a fundamental design principle that ensures both system security and stability. User space applications operate under restricted access to hardware, while in kernel space, there is full control over the system resources, including core operations such as process scheduling, memory management, device interaction and handling system calls.

This separation prevents user applications from directly accessing and modifying sensitive system components. Instead, interactions occur through system calls, which serve as controlled interfaces to the kernel. Through these interfaces, applications can request operations like file access, process creation, or network communication, all while remaining subject to strict privilege and access controls.

From a security perspective, the kernel provides a powerful vantage point for observing and enforcing system behaviour. Since all essential operations flow through the kernel, embedding security mechanisms at this layer enables comprehensive monitoring and robust control over system activity. In contrast to user-space solutions, which may be manipulated by malicious processes, kernel-level approaches provide stronger integrity and support real-time enforcement of security policies.

This separation is particularly important for intrusion detection systems. Kernel-level operation enables observation of low-level events such as system calls and network packet processing at their origin. This level of visibility improves detection accuracy and immediate response to potential threats.

The system call interface is the primary mechanism through which user-space applications

interact with the operating system kernel.
System calls provide controlled access to kernel functionalities, enabling processes to perform operations such as file manipulation, process creation, inter-process communication, and network interactions. Acting as a bridge between user-space and kernel space, they ensure that sensitive operations are executed in a secure and regulated manner.
Each monitored system call corresponds to a specific kernel function, such as process execution, file operations, network communication. Because these calls are required for virtually all meaningful system activity, they form a crucial observation point for understanding process behaviour and system dynamics.
For intrusion detection systems, this architectural separation is especially valuable. Operating within the kernel allows direct observation of low-level events, including system calls and raw network packet processing. Access to this detailed, real-time information improves the accuracy of detecting abnormal or suspicious activities and enables faster responses to emerging threats [11].
Furthermore, the system call interface provides a powerful enforcement point. By intercepting or observing these calls within the kernel, it becomes possible not only to monitor system activity but also to control it in real time [12]. This is particularly valuable for implementing defences that require immediate action, such as blocking unauthorized operations or validating sensitive actions before they are carried out.
As a result, system calls form a foundational component of kernel-level intrusion detection systems, providing both fine-grained visibility and precise control over system behaviour.

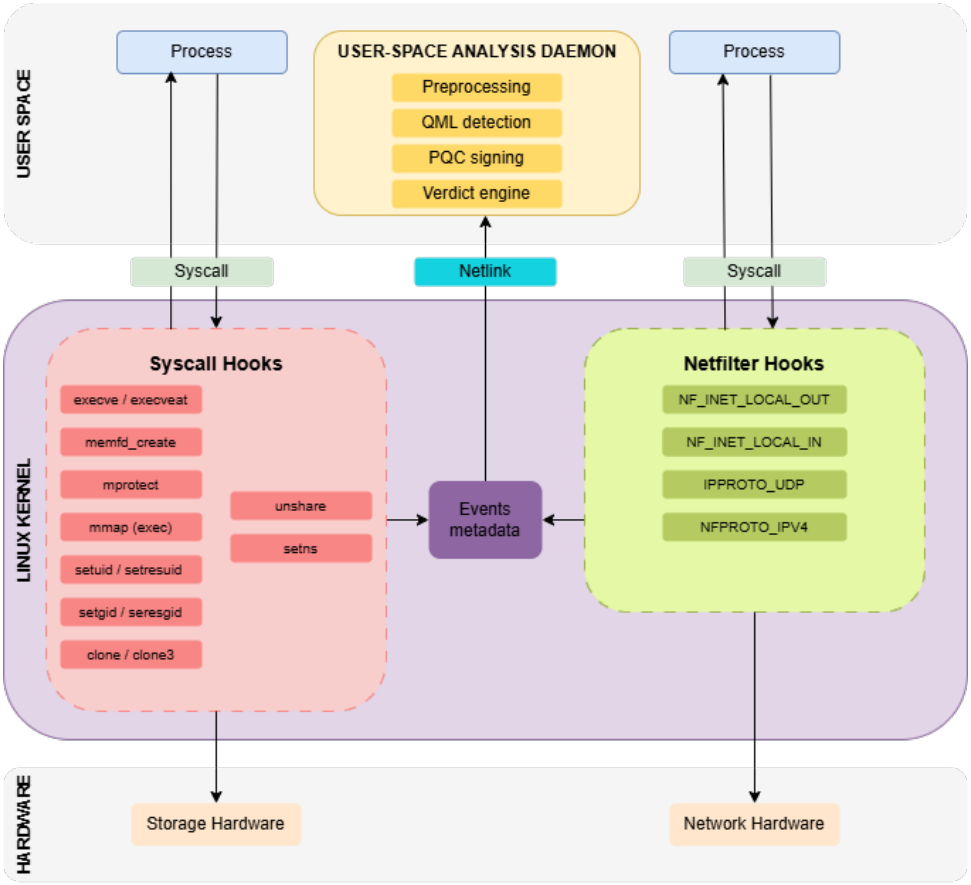

Figure 1. Linux Kernel Components Used in the Proposed Architecture

Monitoring system calls at the kernel level provides highly detailed insight into process behavior and system activity. Unlike user-space monitoring, which can be bypassed or manipulated, kernel-level interception ensures that all critical operations are observed

at their source. Linux offers several mechanisms for observing system calls, including **kprobes**, **tracepoints**, and **Linux Security Module (LSM) hooks**. Among these, kprobes provide a flexible mechanism for dynamic kernel instrumentation without the need to recompile the kernel itself.

In the proposed system, syscall interception is implemented using kprobes, allowing the monitoring of selected high-impact system calls related to process execution, memory operations, and privilege management. These calls are chosen based on their association with common attack techniques, including malware execution, privilege escalation, and container or namespace manipulation.

As it can be seen from the figure above, process execution is monitored through **execve** and **execveat**, which are triggered whenever a new program is launched. These calls reveal which binaries are being executed and serve as a primary indicator for detecting unauthorized or suspicious process activity. Additionally, **memfd_create** is monitored to identify fileless malware techniques, where malicious payloads are loaded and executed directly from memory rather than being written to disk.

Memory-related system calls such as **mprotect** and **mmap** are also intercepted, especially in cases where memory regions are assigned executable permissions. These events are commonly linked to exploitation techniques, including code injection and runtime payload execution. Monitoring these calls enables the detection of abnormal memory behavior that may indicate an ongoing attack.

Privilege escalation attempts are tracked through system calls such as **setuid**, **setgid**, **setresuid**, **setresgid**, and **capset**. These calls are critical for identifying unauthorized privilege changes, container escapes, or rootkit activity. By observing these transitions, the system can detect deviations from expected privilege usage patterns.

Process and namespace manipulation are further monitored using **clone** and **clone3**, which are responsible for process creation and are frequently used in containerized environments. Additional calls such as **unshare** and **setns** are intercepted to detect namespace isolation changes, which may indicate attempts to evade security boundaries or manipulate container contexts.

By combining these syscall interception points, the system constructs a comprehensive view of process behavior at runtime. Each captured event is transformed into structured metadata and forwarded to the user-space analysis daemon for further processing. This approach enables the detection of complex, multi-stage attack patterns that span numerous system activities, while retaining real-time monitoring capabilities within the kernel.

Table 1. Hook on memfd_create to monitor malware fileless execution

```
static struct kprobe kp_memfd_create = { .symbol_name = "__x64_sys_memfd_create" };

static int handler_pre_memfd(struct kprobe *p, struct pt_regs *regs)
{
    struct qks_event_msg msg;
    const char __user *uname = (const char __user *)ARG0(regs);
    __u64 flags = (__u64)ARG1(regs);

    qks_fill_common(&msg, QKS_SC_MEMFD_CREATE, __NR_memfd_create);
    msg.sc_flags = flags;
    if (uname)
        qks_copy_user_str(msg.sc_str, sizeof(msg.sc_str), uname);

    qks_send_msg(&msg);
    return 0;
}
```

In addition to system call monitoring, gaining visibility into network activity is essential for detecting malicious activity such as command-and-control (C2) communication, data exfiltration, and lateral movement [1]. To achieve this, the proposed system incorporates in-kernel packet inspection using **Netfilter**, a Linux framework that enables interception and manipulation of network packets at various stages of the networking stack.

Netfilter exposes several hook points that allow kernel modules to observe or modify packets as they move through different stages of processing. In this work, hooks such as **NF_INET_LOCAL_OUT** and **NF_INET_LOCAL_IN** are used to monitor outbound and inbound traffic, respectively. Observing both directions provides a complete picture of network behavior, capturing connections initiated by local processes as well as communication coming from external sources.

A core feature of the system is the correlation of network activity with process-level context. For each captured packet, the system associates relevant metadata such as process identifier (PID), executable path, protocol type, destination address, and port information. This enables the identification of suspicious behavior not only based on traffic patterns, but also on the originating process. For example, an unexpected outbound connection from a non-network-oriented process may be an early indicator of malicious activity.

Special attention is given to outbound TCP SYN packets, which mark the beginning of a new connection. Monitoring these events helps identify behaviors such as port scanning, unauthorized external communication, or malware beaconing. Additionally, DNS traffic is analyzed by extracting query names and associating them with the originating process. This provides valuable insight into potential connections with malicious or algorithmically generated domains frequently used by advanced threats.

By combining network-level monitoring with process-aware context, the system improves its ability to detect early-stage attack indicators. Furthermore, linking execution events with subsequent network actions makes it possible to trace suspicious connections back to the binaries that initiated them, enabling more precise detection and more effective response. Once captured, network events are transformed into structured metadata and merged with system call data, producing a unified event representation. These events are then forwarded to the user-space analysis daemon for further processing and decision-making.

Table 2. Netlink Message Construction and Multicast Broadcast

```
skb = genlmsg_new(NLMSG_GOODSIZE, GFP_ATOMIC);
if (!skb)
    return -ENOMEM;

hdr = genlmsg_put(skb, 0, 0, &qks_family, 0, QKS_CMD_EVENT);
if (!hdr) {
    nlmsg_free(skb);
    return -ENOMEM;
}
ret = nla_put(skb, QKS_ATTR_MSG, sizeof(*msg), msg);
if (ret) {
    nlmsg_free(skb);
    return ret;
}
genlmsg_end(skb, hdr);

ret = genlmsg_multicast(&qks_family, skb, 0, 0, GFP_ATOMIC);
```

To support communication between kernel space and user space, the proposed system utilizes Netlink [12], a socket-based inter-process communication mechanism specifically designed for interaction between the Linux kernel and user-space applications. Netlink enables efficient, asynchronous message exchange, making it well-suited for security systems that require real-time event reporting and decision feedback [11].

In this architecture, Netlink serves as the channel through which structured event metadata is transmitted from the kernel module to the user-space analysis daemon. Each captured event, originating from either system call interception or network monitoring, is encapsulated into a predefined message format and sent to the daemon for further analysis. After processing the event, the daemon returns a corresponding verdict, which includes authorization data such as cryptographic signatures.

This bidirectional communication model ensures a clear separation of responsibilities: the kernel is responsible for event collection and enforcement, while the user-space daemon performs computationally intensive operations such as anomaly detection and cryptographic processing [12].

A significant design challenge arises from the constraints of kernel-space programming, particularly when integrating post-quantum cryptographic mechanisms. Unlike user-space, the Linux kernel cannot rely on common cryptographic libraries such as OpenSSL, nor can it use dynamic, memory-intensive frameworks. Additionally, kernel code must adhere to strict limitations regarding memory allocation, execution time, and overall reliability, as any instability may compromise the entire system.

.

Table 3. Joining a Generic Netlink Multicast Group in the User-Space Daemon

```
int grp_id = genl_ctrl_resolve_grp(sk, QKS_GENL_FAMILY, QKS_GENL_MCGRP);
if (grp_id < 0) {
   fprintf(stderr, "[DAEMON] resolve group failed: %s\n", nl_geterror(grp_id));
   return 1;
}

rc = nl_socket_add_membership(sk, grp_id);
if (rc != 0) {
   fprintf(stderr, "[DAEMON] add_membership failed: %s\n", nl_geterror(rc));
   return 1;
} else {
   printf("[DAEMON] joined multicast group OK\n");
}

nl_socket_add_membership(sk, grp_id);
```

Because of these limitations, implementing full post-quantum cryptographic operations directly within the kernel is impractical.

In this work, signature generation is performed in user space using the analysis daemon, where private keys can be securely managed and complex cryptographic computations can be executed without kernel restrictions. The kernel is responsible only for lightweight verification, such as validating message integrity, checking signature structure, and ensuring that received authorization data matches expected formats.

This design reduces the computational burden on the kernel while preserving strong security guarantees. By offloading heavy cryptographic operations to user space and limiting kernel responsibilities to validation and enforcement, the system achieves a balanced architecture that maintains performance, safety, and cryptographic robustness

3 Post-Quantum Cryptographic Integration

The rapid advancement of quantum computing poses a significant threat to classical cryptographic algorithms, particularly those based on integer factorization and discrete logarithms. Algorithms such as RSA and elliptic-curve-cryptography (ECC), which form

the backbone of modern authentication, confidentiality, and integrity mechanisms, are expected to become vulnerable to quantum attacks, most notably through Shor's algorithm. As a result, there is a growing need for security mechanisms that remain resilient in the presence of quantum-capable adversaries.

In the context of intrusion detection systems, cryptographic mechanisms are typically used to ensure data integrity, authentication, and secure communication. However, most existing IDS architectures do not integrate quantum-resistant cryptography, exposed under future threat models that assume adversaries with access to quantum computing capabilities. Furthermore, traditional IDS solutions often focus solely on behavioral detection, without enforcing strong guarantees regarding the authenticity of system actions.

To address these limitations, the proposed system integrates PQC techniques into the decision-making pipeline. By combining anomaly detection with cryptographic authorization, the system adopts a zero-trust model in which system actions must be both behaviorally valid and cryptographically authenticated before execution is permitted.

The system employs ML-DSA-44, a post-quantum digital signature scheme based on lattice cryptography. ML-DSA, also known as the Module-Lattice Digital Signature Algorithm, is derived from the CRYSTALS-Dilithium family and is designed to provide strong resistance against quantum-enabled attacks while maintaining practical performance for real-world deployments [12]. ML-DSA-44 is chosen for its balanced trade-off between security level, signature size, and computational overhead.

The ML-DSA-44 parameter set offers moderate security with reduced key and signature sizes compared to higher security variants, making it suitable for systems where performance and resource constraints are crucial. This is particularly relevant in the context of kernel-integrated security mechanisms, where minimizing overhead is essential to preserve system responsiveness and stability.

The implementation of ML-DSA-44 in this work is based on the PQClean library, an open-source collection of clean, portable implementations of post-quantum cryptographic algorithms. PQClean provides standardized and well-tested implementations, allowing integration into user-space without introducing unnecessary complexity or dependency overhead [7][9].

Due to the constraints of kernel-space programming, all cryptographic operations involving private keys are performed within the user-space analysis daemon. When an event is classified, the daemon generates a digital signature over a canonical representation of the event data.

Specifically, the event metadata is serialized into a consistent, deterministic format and hashed using a cryptographic hash function. The resulting digest is then signed using the ML-DSA-44 private key. This ensures that the signature uniquely corresponds to the observed event, preventing replay attacks or tampering.

Performing signature generation in user space offers several advantages: access to complete cryptographic libraries, flexible memory management, and secure key-handling mechanisms. This design also ensures that private keys never enter kernel space, reducing the risk of key exposure or misuse.

In contrast to signature generation, the system performs full signature verification within the kernel. This ensures that enforcement decisions remain independent of user-space trust assumptions, consistent with a zero-trust architecture in which the kernel acts as the final authority on whether an action is permitted.

When a verdict message arrives from the user-space daemon via Netlink, the kernel extracts the included event metadata, specifically the event hash and the post-quantum signature. Before verification, the kernel recomputes the hash of the event based on the same canonical representation to ensure integrity and prevent tampering during transmission.

The ML-DSA-44 verification algorithm is then applied using the public key stored securely within the kernel. If the signature is valid and matches the recomputed hash, the event is considered cryptographically authenticated. Otherwise, the event is treated as un-

authorized, and the corresponding action is denied.

Implementing full post-quantum signature verification in kernel space introduces several challenges. The Linux kernel environment lacks support for many standard libraries and abstractions typically used in cryptographic implementations, requiring careful adaptation of external code. In addition, kernel constraints such as limited stack size, restricted memory allocation mechanisms, and the absence of floating-point operations further complicate the integration of computationally intensive algorithms like ML-DSA.

To overcome these limitations, the verification logic is incorporated in a minimal and controlled manner, focusing strictly on the required functionality. Memory usage is carefully managed, and only essential components of the algorithm are included to avoid unnecessary overhead. Despite these constraints, the system successfully performs full post-quantum signature verification within the kernel, enabling strong authenticity guarantees without relying on user-space trust.

This design significantly enhances security: even if the user-space daemon is compromised, unauthorized actions cannot be executed without a valid post-quantum signature recognized by the kernel.

The proposed system implements a complete authorization pipeline that integrates kernel-level monitoring, user-space analysis, and post-quantum cryptographic verification. This pipeline ensures that every monitored action is evaluated both behaviorally and cryptographically before being allowed or denied.

The process begins in the kernel, where system calls and network events are intercepted using kprobes and Netfilter hooks. Each event is transformed into a structured metadata representation and transmitted to the user-space analysis daemon via Netlink [10][11].

Upon receiving the event, the daemon performs preprocessing and behavioral analysis using quantum-inspired machine learning techniques. Based on this analysis, the event is classified as normal or anomalous. For events deemed legitimate, the daemon generates a cryptographic signature using the ML-DSA-44 private key, applied to a hash of the event data.

The daemon then sends a response back to the kernel, containing the original hash, the generated signature, and the corresponding verdict (e.g., allow or deny). When the response is received, the kernel recomputes the hash of the original event and verifies the signature using the stored public key.

If both the behavioral classification and cryptographic verification succeed, the kernel allows the operation to proceed. If either check fails, due to anomalous behavior or invalid signature, the kernel blocks the action and logs it for further analysis.

This dual-layer validation model ensures resilience against both previously unseen behavioral threats and unauthorized or tampered actions. By combining real-time monitoring, intelligent analysis, and post-quantum cryptographic enforcement, the proposed architecture achieves a robust, scalable and future-proof security mechanism suitable for modern operating systems.

4 Quantum-Inspired Machine Learning Component

Existing rule-based detection mechanisms are effective against known attack patterns but struggle to identify new, unseen, and evolving threats. To add more to the deterministic policy engine, the proposed architecture incorporates a quantum-inspired machine learning (QML) component that can analyses behavioral patterns that do not match any predefined rule.

The QML model acts as a second-stage analysis mechanism, that is used after checking the events with the policy engine. If an event matches a known malicious pattern the verdict is generated immediately, but if not, it is sent to the machine-learning layer. On this layer, rather than analyzing each individual event, the daemon aggregates process activity into short temporal windows of approximately one second. For each process, both syscall and network activity are collected and transformed into a feature representation that describes the recent behavior of that process.

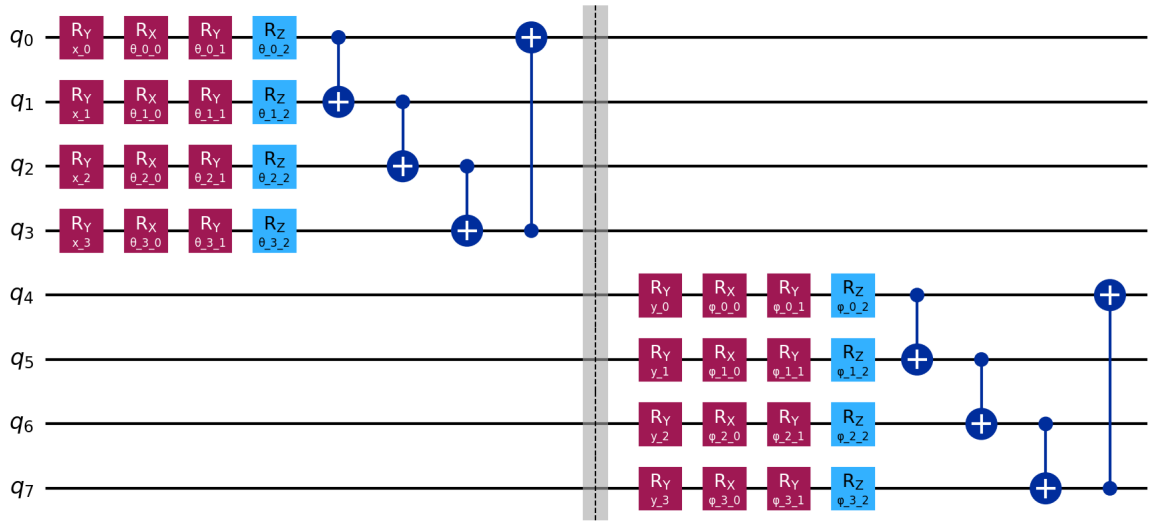

Figure 2. Dual-Encoder architecture

For this, a Dual-Encoder model that processes syscall features and network features using separate variational quantum circuits was used. Feature values are encoded into quantum states using angle embedding, where normalized inputs determine the rotation applied to quantum gates[13]. The encoded states are processed through multiple variational layers composed of trainable rotation gates and entangling operations. During training, these parameters were optimized to achieve a better accuracy. The final circuit measurements are transformed into anomaly scores that indicate the likelihood that the respective behavior derivates from expected patterns [14].

Experimental evaluation showed that the method obtaining an F1-score of 0.775 and an AUC of 0.807. These results do not indicate perfect detection accuracy, but they show that the model can identify meaningful behavioral anomalies that are not captured by the static rules.

In the final system architecture, the QML component does not replace traditional policy-based detection, but instead complements it. This hybrid approach combines the explainability of machine learning, providing an additional layer of protection against evolving threats while preserving the overall responsiveness of the system.

5 Experimental validations

The proposed intrusion detection system was evaluated in a controlled Ubuntu 24.04 environment to validate the integration of kernel-level monitoring, policy-based decision making, quantum-inspired machine learning, and post-quantum cryptographic verification.

The experimental results confirmed that rule-based detection operated as expected. Events matching predefined malicious patterns generated immediate verdicts, while events that could not be classified through policy evaluation were forwarded to the machine-learning component for further analysis.

To assess the practical impact of this architecture, memory consumption was measured for the primary user-space components during normal operation.

Table 3. Memory Consumption of User-Space Components

Component	Resident Memory (RSS)
QKS daemon	5.6 MB
ML service	367.3 MB

The results indicate that the monitoring and decision-making daemon introduces only a modest memory overhead, requiring approximately 5.6 MB. The machine-learning service used a larger memory of approximately 367 MB. This increase is primarily coming from

the Python runtime environment, numerical processing libraries, and the quantum-circuit simulation framework used to execute the variational quantum models. Despite this additional overhead, the combined memory consumption remained below 400 MB, demonstrating that the proposed architecture can operate on commodity hardware without excessive resource requirements.

6 Conclusions

This paper presented a hybrid intrusion detection and firewall system that integrates kernel-level monitoring with post-quantum cryptographic verification and quantum-inspired behavioral analysis. By combining system call interception and network packet inspection within the Linux kernel with intelligent user-space processing, the proposed architecture achieves comprehensive visibility over system activity while maintaining real-time enforcement capabilities.

A key contribution of this work is the integration of post-quantum cryptographic mechanisms into the decision-making pipeline of an intrusion detection system. By employing the ML-DSA-44 signature scheme and performing full signature verification within the kernel, the system enforces a zero-trust model in which actions must be both behaviorally legitimate and cryptographically authenticated. This ensures resilience not only against current vectors but also against future adversaries equipped with quantum computing capabilities.

The design also emphasizes a clear separation of responsibilities between kernel and user space. Computationally intensive operations, such as anomaly detection and signature generation, are executed in user space, while the kernel focuses on event collection, lightweight validation and final enforcement. This division achieves a practical balance between performance, reliability, and security, addressing the inherent constraints of kernel-level development.

Despite its advantages, the proposed architecture presents several challenges. Integrating post-quantum signature verification into the kernel requires careful adaptation of external code due to limited library support and strict execution constraints of kernel environments. Additionally, advanced behavioral detection techniques may introduce performance overhead, particularly under heavy system load or in environments with high event frequency.

Future work may focus on optimizing the performance of kernel-level verification, exploring hardware acceleration for post-quantum cryptographic operations. Additional research may extend the system to cover a broader range of attack scenarios, incorporate more advanced quantum machine learning models, or evaluate the architecture in large-scale or production environments [11].

Overall, this work demonstrates that combining kernel-level monitoring with post-quantum cryptographic enforcement and intelligent analysis is a viable foundation for next-generation intrusion detection systems. As cybersecurity continues to evolve in response to emerging technologies, such hybrid architectures offer a promising pathway toward building robust, scalable, and future-proof system protection mechanisms.

References

[1] H. J. Liao, C. H. R. Lin et al., “Intrusion detection system: A comprehensive review” *Journal of Network and Computer Applications*, vol. 36, p. 16-24, 2013.

[2] Byung-joo Kim and Il-kon Kim, "Kernel based intrusion detection system," *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, Jeju Island, South Korea, 2005, pp. 13-18, doi: 10.1109/ICIS.2005.78.

[3] “Improving Host Security with System Call Policies”, *12th USENIX Security Symposium (USENIX Security 03),* 2003.

[4] “IMA and EVM Concepts”, [Online]. Available: https://ima-doc.readthedocs.io/en/latest/ima-concepts.html#ima-and-evm-concepts

[5] Hofmeyr SA, Forrest S, Somayaji A. “Intrusion detection using sequences of system calls”, *Journal of Computer Security*, vol. 6, p. 51-180, 1998, doi:10.3233/JCS-980109.

[6] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," in *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278-1308, Sept. 1975, doi: 10.1109/PROC.1975.9939.
[7] Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, and Thom Wiggers. "Improving Software Quality in Cryptography Standardization Projects.", *Security Standardization Research – EuroS&P Workshops*, 2022.
[8] NIST FIPS 2024, "Module-Lattice-Based Digital Signature Standard (ML-DSA)" 2024.
[9] Matthias J. Kannwischer et al., "PQClean: Clean and Auditable Implementations of Post-Quantum Cryptography", Journal of Open Source Software, 2022
[10] Salim, J., et al. "Linux netlink as an ip services protocol*",* no. rfc3549. 2003.
[11] P. B. S. L. Team, "Netlink Sockets Overview," *Linux Kernel Documentation*, 2023. [Online]. Available: https://www.kernel.org/doc/html/latest/userspace-api/netlink/index.html.
[12] NIST IR 8547, "Transition to Post-Quantum Cryptography Standards" National Institute of Standards and Technology, 2025.
[13] Maria Schuld , Francesco Petruccione, "Machine Learning with Quantum Computers", Springer, 2021
[14] M. Schuld, A. Bocharov, K. Svore and N. Wiebe, "Circuit-Centric Quantum Classifiers", Physical Review A, 2020

Mariuca PRICOP has a bachelor's degree in Economic Informatics and an ongoing MSc. degree in Cyber Security from the Department of Economic Informatics and Cybernetics at Bucharest University of Economic Studies, Romania. Her research interests are Systems Security, Intrusion Detection Systems, Antivirus and Antimalware, Post-Quantum Cryptography, and AI-Driven Cybersecurity.

Cristian TOMA is professor at Department of Economic Informatics and Cybernetics, and he has graduated the Economic Informatics bachelor program, within University of Economic Studies Bucharest in 2003. He has graduated from the BRIE master program in 2005, with practical stage in Germany and the PhD stage in 2008. He is cofounder of the IT&C | Cyber Security Master Program (www.ism.ase.ro) and cofounder of the SECITC – The International Conference on Security for Information Technology and Communications (www.secitc.eu). His work focuses on cybersecurity in IoT - Internet of Things, Crypto Blockchain, embedded/Secure Elements, Cloud/High Performance Computing, Computer antiviruses, Robotics, Artificial Intelligence, Quantum Computing, and Post Quantum Cryptography.