ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 10, No. 1, 2025

ENHANCING HOST INTRUSION DETECTION SYSTEMS FOR LINUX BASED NETWORK OPERATING SYSTEMS

Bohdan Havano, Andriy Dobush

Lviv Polytechnic National University, 12, S. Bandery str., Lviv, 79013, Ukraine
Authors' e-mails: *bohdan.i.havano@lpnu.ua, andrii.r.dobush@lpnu.ua*

https://doi.org/10.23939/acps2025.01.054

Submitted on 24.03.2025

***Abstract*: This paper proposes an enhanced model of Host Intrusion Detection Systems (HIDS) adapted for Linux-based Network Operating Systems (NOS), specifically SONiC. The SONiC architecture has been analyzed to identify intrusion-sensitive components, including telemetry data, container logs, and inter-container communications. A machine learning-based HIDS profile has been introduced to detect anomalies within containerized services and network modules. Signature-based, anomaly-based, and hybrid-based detection approaches have been classified with consideration of NOS-specific traits. The proposed solution has integrated external threat intelligence and adversarial modeling to improve detection accuracy. Results confirm the effectiveness of the method in securing cloud-scale networks powered by open-source NOS platforms.**

***Index terms*: Host intrusion detection, Linux, Network OS, Security.**

I. INTRODUCTION

Intrusion detection involves monitoring and analyzing events within a computer system or network to identify potential security breaches. These threats can range from malware and DoS / DDoS attacks to unauthorized access, privilege escalation, or probe attacks. While many events that seem malicious are genuine attacks, there are exceptions, such as users mistyping an address or unintentionally connecting to the wrong system. It's crucial for the system to accurately differentiate between actual intrusions and normal network traffic. Ultimately, an Intrusion Detection System (IDS) automates the process of identifying and responding to such attacks [1–3].

In numerous studies, Host Intrusion Detection Systems (HIDS) are commonly applied to general-purpose Linux hosts, such as servers and network-connected user devices. In today's rapidly evolving cybersecurity landscape, HIDS play a crucial role in monitoring the integrity and security of individual computing systems. Typically designed to operate within a single host, HIDS focus on analyzing internal activities to detect and prevent potential threats [4–6]. With various types of HIDS tailored to specific environments and needs, a deeper understanding of their capabilities is essential.

HIDS solutions range from signature-based detection to advanced anomaly detection mechanisms. While some systems rely on predefined rules to identify suspicious behavior, others use machine learning to detect anomalies in real time. Whether rule-based or dynamic, HIDS are essential in securing devices such as servers, workstations, and mobile devices against cyber threats.

Traditionally, HIDS monitor events like file system changes, process execution, and network connections on the host machine to identify unauthorized access, malware, and other malicious activities. However, as the computing paradigm shifts towards interconnected systems driven by Network Operating Systems (NOS), traditional HIDS may not be as effective in protecting these complex environments [7].

Given this shift, there is a growing need to extend HIDS capabilities to better integrate with Network OS environments. Enhancing HIDS to account for network communications, protocol analysis, and distributed threat detection can strengthen the security of interconnected systems. This article examines the need for adapting HIDS to work within Network OS frameworks, exploring the challenges and opportunities in improving these essential security tools to meet the demands of an increasingly interconnected world.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

Host based IDS is aimed at collecting and analyzing information on a particular host or system. Such agent monitors and prevents intruders to compromise system security policy. Comparing to Anti-virus HIDS play different role. As it has been mentioned in [1] HIDS check and collect system data including File System, Network Events and System Calls to verify whether any inconsistency has occurred or not. HIDS relies on audit trail and system logs to detect unusual activities inside the system.

Several critical factors influence effective decision-making in intrusion detection systems (IDS) are enumerated in [7]. The IDS must remain operational and resilient during failures or high-load scenarios to ensure uninterrupted threat monitoring. Timely identification of threats is essential to prevent the spread of attacks within interconnected NOS modules. Reducing false alerts helps maintain the system's credibility and ensures that real threats receive prompt attention. The IDS should accurately detect a broad range of attacks, including novel or zero-day threats, to ensure comprehensive protection.

Efficient resource consumption is necessary to avoid impacting the performance of SONiC switches and network operations. The IDS must pinpoint the exact location of suspicious activity, whether in containers, configuration files, or Redis interactions. Lastly, the system should seamlessly integrate with external security tools and intelligence feeds for enhanced threat analysis and response.

In summary, an IDS must provide the above-mentioned features for high accuracy and timely detection of attacks.

According to [3] intrusion detection systems (IDSs) typically consist of three primary elements:

1. *Data collection*: acquire various types of data, such as system calls or network flows.
2. *Feature extraction*: This involves transforming a predetermined unit of data, like system calls within a process or flows within a time frame, into a set of attributes known as a feature vector.
3. *Decision mechanism*: This component employs an algorithm or heuristic to determine whether the provided data, in its feature vector form, is indicative of an attack or not.

The decision mechanism can be classified as either misuse, anomaly, or a hybrid detector. Misuse intrusion detection relies on pre-established attack patterns, such as signatures of known malware or expertly crafted rules, to identify matching events. As it has been mentioned in [3] zero-day attacks, which are novel or exploit previously undiscovered vulnerabilities, often bypass misuse detection algorithms. Typically, misuse detection relies on a database of attack signatures that tends to be large, continuously expanding, potentially challenging to employ effectively, and requires frequent updates. Having predefined signatures of attack, misuse detection typically has a relatively low rate of false positives but a high rate of false negatives.

In anomaly detection, expected behavior is learned from observations of common host work. Any significant deviation from this established profile is identified as a potential attack. Such approach allows detection of previously unseen attack patterns. As it has been described in [4] anomaly detection systems that update in almost real-time can adapt models to the gradually shifting system dynamics. However, the main drawback of anomaly detection lies in its detection accuracy, particularly in its tendency to produce higher rates of false alarms. Furthermore, attacks may camouflage themselves within the background noise of ambient data if the training data, used to establish normal behavior, exhibits significant variation. Likewise, if attacks are present within the training data, detectors may learn to consider such behavior as normal. In [8] author proposed Ab-HIDS, an anomaly-based host intrusion detection system that utilizes N-gram frequency analysis of system calls combined with ensemble learning techniques to effectively detect malicious activity in containerized Linux environments, demonstrating improved accuracy in behavioral anomaly detection models.

As it has been mentioned in [2], intrusion detection systems can rely on several key behavioral measures to identify potential threats. One important category is login and session activity, which encompasses metrics such as login frequency, the distribution of login attempts across different positions, the duration of each session, and the output of executed commands. Another crucial aspect is resource utilization, which includes indicators like the number of failed login attempts, the execution of specific commands and procedures, operational frequency, and the extent of system resource usage. Additionally, file operation activity serves as a valuable measure, focusing on the frequency of actions such as reading, writing, creating, and deleting files.

Host Intrusion Detection Systems (HIDS) can be broadly categorized based on the method they use to detect intrusions as it has been described in Table. The two principal categories are Misuse-based Detection and Anomaly-based Detection, with some modern systems adopting Hybrid approaches. Each type has distinct operational characteristics, advantages, and limitations. The selection of the most appropriate HIDS type depends on the deployment context – especially in environments like SONiC-based NOS where modular, real-time analysis is essential.

Comparative Analysis of HIDS Detection Approaches

Criteria	Misuse-based detection	Anomaly-based detection	Hybrid detection
Detection Method	Pattern matching with known attack signatures	Identifies deviations from established normal behavior	Combines signature and anomaly methods
Accuracy	High precision for known attacks; low false positives	Can detect novel attacks but higher false positives	Balances precision and recall
Zero-Day Attack Detection	Limited (cannot detect unknown threats)	Capable (learns new behaviors)	Strong (leverages both detection paths)
Performance overhead	Low to moderate (dependent on signature DB size)	High (real-time statistical / ML computations)	Moderate to high (depending on implementation)
Data requirement	Signature database	Clean and complete training data	Requires both: clean baseline + updated signature sets

III. SCOPE OF WORK AND OBJECTIVES

This paper investigates the types and methodologies of Host Intrusion Detection Systems (HIDS), focusing on their relevance and adaptability within Network Operating Systems (NOS). The study explores the specific challenges of applying HIDS in the context of modular,

container-based NOS environments, with SONiC selected as a representative open-source example. It examines how traditional HIDS approaches such as signature-based, anomaly-based, and hybrid detection align with the architectural and operational demands of modern NOS platforms. Particular attention is given to identifying which components and data sources within SONiC, are most suitable for intrusion monitoring. The research further outlines the key considerations for integrating HIDS into open-source NOS architectures and presents a conceptual framework to support this integration. Ultimately, the work aims to clarify the requirements and opportunities for enhancing security in disaggregated network systems through tailored HIDS strategies.

IV. NETWORK OS ARCHITECTURE

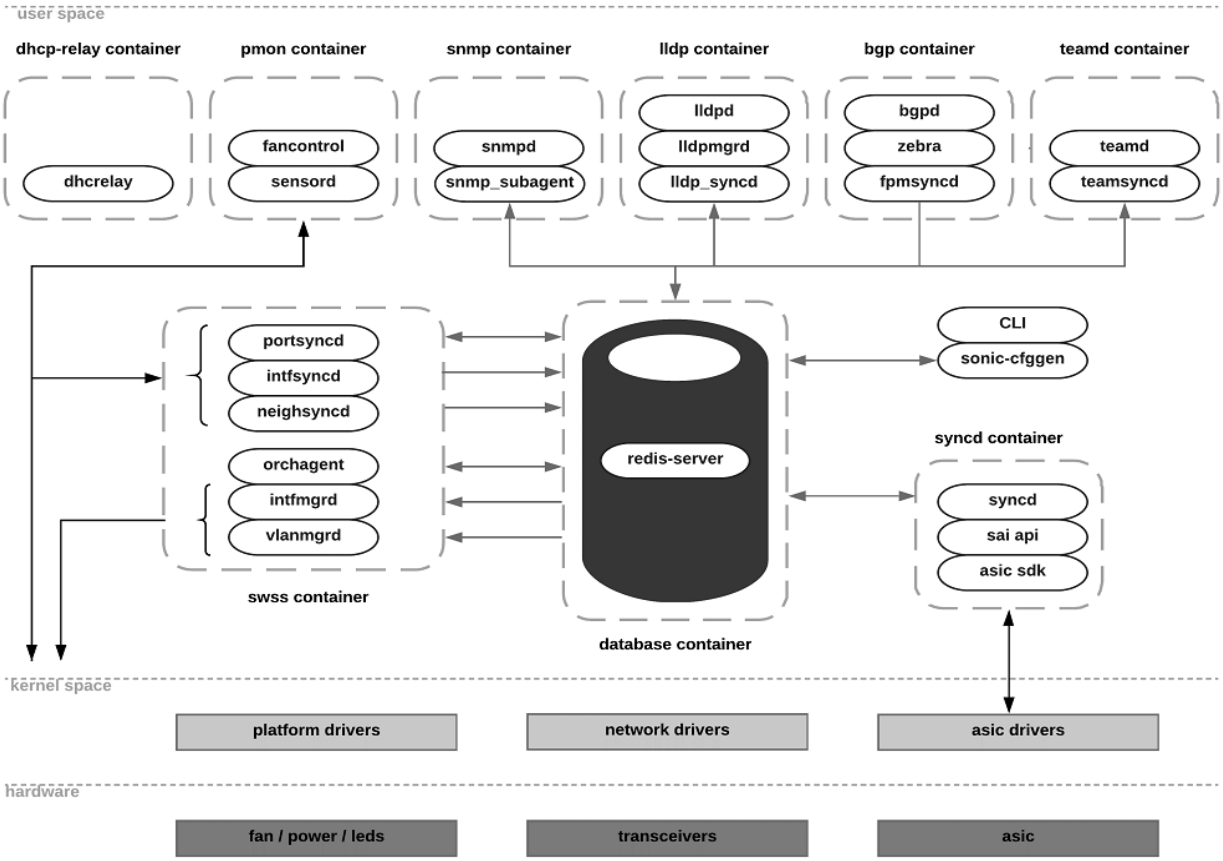
Since the beginning of the networking industry, routing and switching devices have been confined to tightly integrated hardware and software components.

B Manufacturers offer closed-source proprietary stacks, limiting network operators from employing customized features, as a result impending innovation. This bundled approach proves expensive, time-consuming, and non-scalable, as device modifications necessitate vendor intervention. Consequently, the industry has embraced the production of white-box switches and the development of Network Operating Systems (NOSs) that accommodate multiple vendors and Application Specific Integrated Circuits (ASICs). This approach, known as "disaggregated", involves the separation of software and hardware, allowing vendors' switching silicon (e. g., Broadcom) to be compatible with various NOS.

The range of available Network Operating Systems (NOS) is expanding quickly as a result of a competition-driven innovation race which enables organizations to choose the best option to match their use case. Classic, proprietary NOS solutions come with license fees but promise reliable support and new features to be added by the vendor. With this approach, users rely on the vendor's roadmap for innovation within the product and have a little to no control over its functionality.

The separation of hardware and software in white-box switches has accelerated the development and maintenance of open-source network operating systems (NOS). As a result, production-ready solutions like SONiC (Software for Open Networking in the Cloud) and the Dent OS have gained traction in the NOS market. By adopting a whitebox model with an open-source NOS, organizations can take advantage of network disaggregation helping lower expenditures on system integration and eliminate costly software licensing fees.

SONiC – Software for Open Networking in the Cloud – represents a classic approach to network



High level SONiC design

operating systems. Created by Microsoft for its Azure data centers in 2016, it is based on Linux distributive Debian and consists of kernel patches, device drivers, utilities, and user space applications. SONiC adopted Docker containers to successfully address NOS component packaging issues. Currently, this NOS runs on nearly all ASICs with switches available from most equipment vendors. In 2022, SONiC was moved to the Linux Foundation – a center of gravity for the larger open-source community, ensuring the next level of its development and support.

SONiC architecture is described in [5]. It is a Linux-based open source NOS developed by Microsoft. It offers a full-suite of network functionality, like BGP and Remote Direct Memory Access (RDMA) and runs on switches from multiple vendors and ASICs

In [6] SONiC was successfully tested not only as data center-oriented NOS but also as NOS that can be widely used in campus network. Although open source NOSs and white-box switches are mainly targeted to large data centers, they have strong potentials to replace closed source proprietary switches used in campus networks.

SONiC consists of several modules that exist either in docker containers or in the Linux-host system itself. A container is a lightweight, standalone, executable package of software that contains the code, runtime, system tools, system libraries, and settings needed to execute the application. The high-level architecture of SONiC is shown in Fig. 1, where it operates in the user space. Each component in SONiC handles a specific job, such as relaying the DHCP requests, handling Link Layer Discovery Protocol (LLDP) functionalities, providing Command Line Interface (CLI) and system configuration capabilities, and running FRR or Quagga routing stacks.

The architecture of the SONiC system consists of various modules that interact through a centralized and scalable infrastructure. This infrastructure relies on a redis-database engine, which is a key-value database used to provide a language-independent interface, ensure data persistence, enable replication, and facilitate multi-process communication among all SONiC subsystems. By leveraging the publisher / subscriber messaging paradigm provided by the redis-engine infrastructure, applications can subscribe only to the data views they need, thus avoiding implementation details that are irrelevant to their functionality.

SONiC adopts a strategy of placing each module in independent docker containers to maintain high cohesion among semantically related components while reducing coupling between disparate ones. Each of these components is designed to be entirely independent of the platform-specific details required to interact with lower-layer abstractions.

Currently, SONiC divides its main functional components into the following docker containers: dhcp-relay, pmon, snmp, lldp, bgp, teamd, database, swss, syncd. Still its possible to extend SONiC functionality using external prepared docker images. Figure 1 displays from a high-level view of the functionality enclosed within each docker-container, and how these containers interact among themselves. The architecture of SONiC is designed to be modular, scalable, and flexible to meet the requirements of modern cloud-scale networks. A high-level overview of its architecture contains of next items: Linux kernel, Switch Abstraction Interface (SAI), SWSS (SONiC Web Service), Database (REDIS), ASIC-specific Components, Platform-specific Components, Containerized Services and Open-source Software Stack. SONiC is built on top of a Linux kernel, leveraging its stability, performance, and vast ecosystem of drivers and tools.

Switch Abstraction Interface (SAI): SONiC utilizes SAI, which is a standardized API (Application Programming Interface) for programming network ASICs (Application-Specific Integrated Circuits). SAI abstracts the underlying hardware-specific functionalities, enabling SONiC to work with a variety of network hardware platforms.

SWSS (SONiC Web Services): SWSS provides a set of REST APIs for managing and configuring network switches. It acts as a bridge between the SONiC software stack and external management applications, allowing operators to interact with SONiC programmatically.

SONiC uses a Redis database to store network configuration and state information. This database is accessed by various components within SONiC to retrieve and update network settings dynamically.

ASIC-specific Components: SONiC includes ASIC-specific components that interface with the underlying hardware ASICs. These components translate high-level network configuration commands into ASIC-specific instructions and handle tasks such as packet forwarding, ACL (Access Control List) processing, and QoS (Quality of Service) enforcement.

Platform-specific Components: SONiC includes platform-specific components responsible for managing hardware resources such as fans, power supplies, and temperature sensors. These components ensure the proper functioning and monitoring of the underlying hardware infrastructure.

Containerized Services: SONiC adopts a containerized approach for running various network services and applications. Services such as routing protocols (e. g., BGP, OSPF) and monitoring tools (e. g., SNMP, telemetry) are packaged as containers, providing isolation, scalability, and ease of deployment.

Open-source Software Stack: SONiC incorporates various open-source software components for networking functionalities, including FRRouting (FRR), Quagga, and OpenSwitch. These software packages provide routing, switching, and other networking capabilities within the SONiC ecosystem.

Overall, SONiC's architecture emphasizes modularity, standardization, and openness, enabling interoperability across diverse hardware platforms and facilitating the development of innovative network solutions for cloud-scale environments. From listed items there are vendor specifics items: SAI, ASIC and Platform specifics, which may vary from vendor to vendor. On other hand common items should be monitored as part of host intrusion

detection system: Containerized Services, Open-source Software Stack, Database, SWSS, SAI, Linux Kernel.

V. SONIC NOS PROFILE FOR HOST INTRUSSION DETECTION SYSTEM

To construct a machine learning (ML) profile tailored for a Host Intrusion Detection System (HIDS) within a SONiC-based architecture, it is critical to leverage telemetry and host-level data sources that conventional HIDS often overlook. The key data categories informing this enhanced profile include: network telemetry, hardware metrics, container activity logs, configuration changes, inter-container communication, Redis database activity, threat intelligence feeds, and adversarial behavior patterns.

Switch hardware metrics like CPU and memory usage, or interface errors, help identify infrastructure-level threats. Container logs from routing daemons or monitoring agents are analyzed for suspicious executions or resource spikes, while configuration change monitoring targets unauthorized edits to ACLs or routing settings.

Inter-container communication is scrutinized for abnormal lateral movement or C2 behavior across containers. The Redis database, a core SONiC component, is observed for irregular queries or tampering. Additionally, integrating external threat intelligence enriches the detection pipeline by correlating SONiC-specific behavior with known indicators of compromise. Lastly, adversarial modeling trains the ML system to recognize evasive techniques that mimic legitimate patterns to bypass detection.

Together, these sources enable the development of a robust, SONiC-aware HIDS capable of intelligent, real-time response to complex threats in cloud-scale environments.

VI. CONCLUSION

Network infrastructures increasingly adopt distributed and modular architectures powered by Network Operating Systems. Traditional HIDS approaches must be adapted to meet new challenges. This article examined the application of host intrusion detection system within network operating systems, with a particular focus on SONiC, an open-source network operating system designed for disaggregated network hardware. The paper analyzed SONiC's modular architecture and proposed an enhanced host intrusion detection systems profiling approach that incorporated telemetry data, container logs, configuration monitoring, and configuration activity. These elements formed the basis for intelligent, machine learning-driven threat detection tailored to modern networked environments. The study highlights the need for integrated, adaptive HIDS solutions capable of addressing the complexity and scale of today's cloud-native systems.

References

[1] Satilmiş, H., Akleylek, S., & Tok, Z. Y. (2024). A systematic literature review on host-based intrusion detection systems. *Ieee Access*, *12*, 27237–27266. DOI: https://doi.org/10.1109/ACCESS.2024.3367004

[2] Liu, M., Xue, Z., Xu, X., Zhong, C., & Chen, J. (2018). Host-based intrusion detection system with system calls: Review and future trends. *ACM computing surveys (CSUR)*, *51*(5), 1-36. DOI: https://doi.org/10.1145/3214304

[3] Ou, Y. J., Lin, Y., & Zhang, Y. (2010, April). The design and implementation of host-based intrusion detection system. In *2010 third international symposium on intelligent information technology and security informatics,* 595–598. IEEE. DOI: https://doi.org/10.1109/IITSI.2010.127

[4] Jose, S., Malathi, D., Reddy, B., & Jayaseeli, D. (2018, April). A survey on anomaly based host intrusion detection system. In *Journal of Physics: Conference Series* (Vol. 1000, p. 012049). IOP Publishing. DOI: https://doi.org/10.1088/1742-6596/1000/1/012049.

[5] SONiC Network OS architecture document. (2025). [Electronic resource]. Available: https://github.com/sonic-net/sonic/wiki/architecture.

[6] AlSabeh, A., Kfoury, E., Crichigno, J., & Bou-Harb, E. (2020, July). Leveraging sonic functionalities in disaggregated network switches. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)* (pp. 457-460). IEEE. DOI: https://doi.org/10.1109/TSP49548.2020.9163508

[7] Ozkan-Okay, M., Samet, R., Aslan, Ö., & Gupta, D. (2021). A comprehensive systematic literature review on intrusion detection systems. *IEEE Access*, *9*, 157727-157760. DOI: https://doi.org/10.1109/ACCESS.2021.3129336.

[8] Joraviya, N., Gohil, B. N., & Rao, U. P. (2024). Ab-HIDS: An anomaly-based host intrusion detection system using frequency of N-gram system call features and ensemble learning for containerized environment. *Concurrency and Computation: Practice and Experience*, *36*(23), e8249. DOI: https://doi.org/10.1002/cpe.8249

Bohdan Havano was born in 1994 in Sambir, Ukraine. He received the B.S. degree in computer engineering at Lviv Polytechnic National University in 2015 and M.S degree in system programming at Lviv Polytechnic National University in 2016. He has been doing scientific and research work since 2017. His research interests include architecture and data protection in cyber-physical systems.

Andriy Dobush obtained his master's degree in computer engineering, specializing in Computer Systems and Networks, at Lviv Polytechnic National University in 2010. Research interests include security in network operating systems.