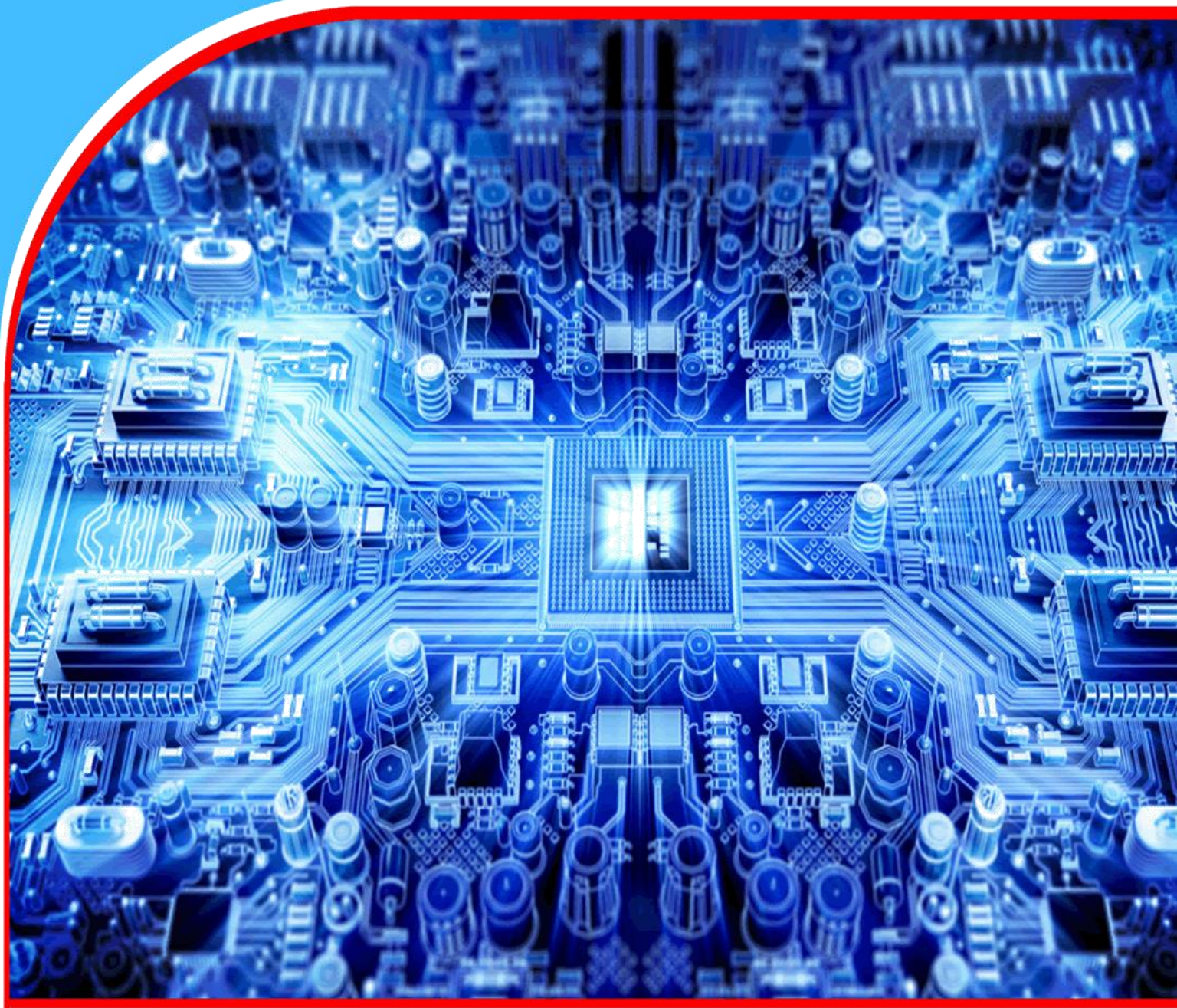
American Journal of
Computing and Engineering
(AJCE)



Fine- Grained Behavioral Analysis for Malware Detection in Containerized Environments

Khaja Kamaluddin

American Journal of Computing and Engineering
ISSN 2790-5586 (Online)
Vol.4, Issue 4, pp 1-20, 2021

www.ajpojournals.org

Fine-Grained Behavioral Analysis for Malware Detection in Containerized Environments

Khaja Kamaluddin

Masters in Sciences, Fairleigh Dickinson University, Teaneck, NJ, USA, Aonsoft International Inc, 1600 Golf Rd, Suite 1270, Rolling Meadows, Illinois, 60008 USA

Article history

Submitted 19.04.2021 Revised Version Received 20.05.2021 Accepted 24.06.2021

Abstract

Purpose: Containerized environments have become foundational to modern software development due to their portability, scalability, and efficient resource utilization. However, their shared-kernel architecture introduces distinct security challenges, particularly in malware detection. This study presents a historical analysis of fine-grained, behavior-based malware detection techniques within containerized systems.

Materials and Methods: We examine early machine learning approaches, including Decision Trees, Hidden Markov Models, and LSTM networks trained with limited datasets alongside system call tracing and process behavior profiling.

Findings: While these techniques are now outdated, they marked critical early steps beyond static and signature-based detection in dynamic, containerized infrastructures. We analyse behavioural features such as syscall sequences, memory anomalies, and DNS irregularities, assessing their detection performance and limitations in orchestrated environments. The paper further contextualizes these legacy methods in light of modern advancements, including eBPF-based monitoring and context-aware deep learning models.

Unique Contribution to Theory, Practice and Policy: Key recommendations include leveraging eBPF for efficient runtime monitoring, incorporating orchestration metadata for context-aware detection, and enabling cross-container correlation for identifying lateral movement. This retrospective establishes a comparative framework that informs the development of adaptive, real-time security solutions, such as graph neural networks and behavioural baselining, thereby guiding future research in runtime container security.

Keywords*: Container Security (O33); Behavioral Malware Detection (D83, O33); eBPF Monitoring (C63, C88); Kubernetes Security (O33); Runtime Threat Detection (C63, O33); Cloud-Native Security (O33, L86); Anomaly Detection (C63, D83); Historical Security Analysis (D83, H56).*

INTRODUCTION

The rise of container technologies such as Docker, LXC and Kubernetes has dramatically transformed the way modern applications are developed, deployed, and scaled [1]. Containers offer a lightweight, efficient, and portable solution, enabling applications to run consistently across diverse environments. Containers streamline deployment, improve scalability and maximise resource utilisation by encapsulating an application and its dependencies into a single unit. Containerization has consequently emerged as a crucial element of contemporary software development and operations, especially in cloud-native settings [2].

However, due to an increased popularity of containers, unique issues regarding security have arisen. At the beginning of the life cycle of the container technologies, issues were raised about the isolation between containers and the containers' host operating systems [3]. Although containers create some isolation, they compete for the same underling host services, exposing them to attacks like container escapes, privilege escalation, and resource contention. The new menaces that surrounded potent containerized systems were not amply handled by the traditional security mechanisms such as the host-based firewalls, intrusion detection systems (IDS), which were originally designed for virtual or physical settings. Container runtimes misconfigurations and issues further worsen these problems, indicating a need for specific security solutions for the container model [4].

The figure 1 represents the development of malware detections technologies with a specific focus on traditional to behavior-based changes. This timeline emphasizes the increased need for dynamic behavior-driven defenses, as the containerized environments started dominating the landscape and becoming increasingly advanced.

Moreover, a comparison of threat surfaces between traditional VM's and containerized landscape is presented in the table 1, in order to point out unique security aspects concerning containerized environments. This comparison demonstrates how the container technologies add new risks and complexities hence requiring addressing of old security paradigms.

Table 1: Comparison of Threat Surfaces Between Traditional Virtual Machines (VMs) and Containerized Environments

Aspect	Traditional Virtual Machines (VMs)	Containerized Environments
Isolation	Strong isolation with separate kernels and virtualized resources.	Weaker isolation, as containers share the same kernel.
Resource Sharing	Fully isolated resources (CPU, memory, disk).	Shared resources at the host OS level, increasing risks.
Security Boundaries	Clear separation between VMs and the hypervisor.	Shared kernel increases attack surface.
Attack Surface	Limited to the VM and hypervisor.	Broader attack surface, including container runtime and shared host OS.
Kernel Vulnerabilities	Vulnerabilities affect only the specific VM's kernel.	Kernel vulnerabilities impact all containers on the same host.
Performance Overhead	Higher overhead due to hardware virtualization.	Lower overhead, as containers run as processes on the host OS.

Behavior-based malware detection methods have become increasingly popular in response to these security flaws [5]. Based on behavior detection, behavior-based detection relies on monitoring system activities and identifying anomalies deviating from the norms as opposed to the former signature-based detection systems that focus on established patterns of malicious activity. Signature-based methods are good for known threats but are not so good at detecting original or innovative threats especially those that are designed to avoid detection systems. However, behavior-based approaches provide flexible and agile defense against evolving threats, particularly against the ones that aim at containerized systems [6]. Behavior-based detection systems can detect an unknown or polymorphic malware and bypass signature-based tools by

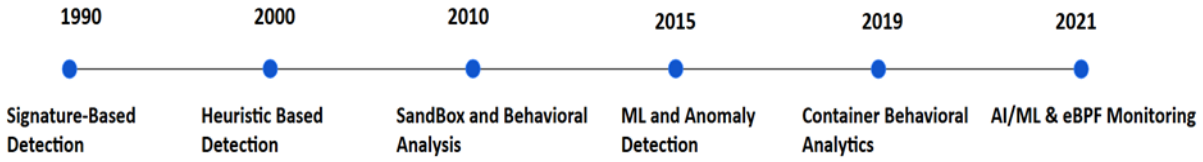


Figure 1: Evolution of Malware Detection Technique

Tracking system calls, usage of resources, and behaviors of processes.

With the containerized environments constantly changing, it has become clear that the high degree of behavioral analysis is required [4]. As opposed to holistic monitoring solutions across the entire system, fine-grained behavioral analysis is aimed at gaining an in-depth overview of the individual containers behavior. Such an approach allows one to reduce false positives in the threat detection systems and make the whole system much more efficient. This approach makes it possible to detect anomalous activity within the individual containers with higher precision [7]. Fine-grained analysis may provide more detailed and accurate insight into the

behavior of containers with the use of knowledge of how files are accessed the use of network communication, and syscall patterns. It is, therefore, an essential tool for modern containerized systems' protection.

This research aims to:

- Assess fine-grained behavioral analysis methods for identifying malwares in containerized environments.
- Find major tools, features and detection models used in the existing container security practices.
- Evaluate the success and constraints of these techniques in solving the modern security challenges.

Background and Motivation

As containerized environments have become central to modern application deployment, they introduce security challenges that demand fine-grained behavioral monitoring from the outset [8]. Unlike virtual machines, containers operate with shared kernels, ephemeral lifecycles, and dynamic orchestration, creating conditions where traditional detection methods fall short [9]. For example, microservice interactions often lead to divergent system call patterns even among containers running the same image rendering host-level detection unreliable. Frequent restarts and scaling events rebuild process trees, requiring continuous tracking to maintain visibility. Additionally, the shared-kernel model introduces noisy neighbor effects, which can mask malicious behavior unless monitoring is done at the per-process, per-container level. These factors highlight why static and signature-based methods are inadequate and underscore the need for behavior-based techniques, such as system call analysis, process lineage tracking, and resource profiling, that operate with the necessary granularity to detect modern threats.

Understanding Behavioral Malware Detection in Containers

Behavioral malware detection is a dynamic approach that identifies threats by analyzing runtime system activities rather than static signatures. Behavioral analysis for malware detection revolves around monitoring system activities to detect abnormal behaviors that may signal the presence of malicious software [10]. Unlike signature-based detection methods, which rely on predefined patterns of known malware, behavioral analysis identifies novel threats by examining the dynamic interactions within the system. This method provides a more flexible and robust way to identify complex or unknown malware by concentrating on patterns of system calls, resource usage, and process behavior. This is especially useful in containerized environments where conventional methods often fail.

Behavioral analysis involves monitoring processes at runtime, observing activities like file access patterns, system call behaviors, network communication, and inter-process communications. This offers a more sophisticated comprehension of how the system functions, making it possible to detect anomalies that might point to malware with greater accuracy.

Principles and Techniques of Behavioral Malware Detection

Key techniques for behavioral malware detection in containerized environments include [11]:

i. System Call Tracing: Capturing system calls made by running processes, enabling the identification of suspicious or malicious behavior by comparing observed actions with normal patterns.
ii. Resource Usage Monitoring: Tracking the use of the CPU, memory, disk, and network to identify resource spikes or irregular patterns indicative of a potential attack.

iii. Process Behavior Analysis: Observing the way processes interact with each other and the host environment, helping to spot abnormal behaviors that could point to malicious activity.

These techniques provide the dynamic capabilities necessary for identifying threats that might evade traditional, static analysis tools.

Key Monitoring Tools and Techniques for Malware Detection

Before the widespread adoption of advanced container security frameworks, several key technologies were crucial for monitoring system behavior and identifying malware:

i. Strace: A system call tracer that helps capture and examine the system calls invoked by processes, enabling detection of suspicious or unauthorized activities.
ii. System Call Tracing: This method monitors system calls as the primary interface between user-space programs and the kernel, allowing security experts to detect unusual interactions indicative of malware.
iii. Auditd: In order to identify irregularities in containerized systems, the Linux Auditing System logs a variety of system operations, such as file access, process execution, and network activity.
iv. Procfs: The proc filesystem exposes kernel data structures to user space, allowing continuous monitoring of system performance and processes. By examining this data, it's possible to detect abnormal system interactions that could signal malicious behavior.

These technologies were fundamental for security monitoring and early malware detection in containerized environments [12]. However, their limitations in handling the scale and dynamic nature of containers became apparent over time. To better understand the evolution of detection strategies, Table 2 provides a comparison between Static, Behavioral, and Hybrid detection approaches in container security.

Table 2: Comparison of Static vs Behavioral vs Hybrid Detection in Containers

Detection Type	Advantages	Disadvantages
Static Analysis	Fast scanning for known signatures. Lower resource usage. Suitable for detecting known threats.	Limited to known malware. Fails to detect zero-day or polymorphic threats. Inadequate for containers.
Behavioral Analysis	Detects unknown or evolving threats. Analyzes real-time activities. Suitable for dynamic environments.	Resource-intensive. Higher false positive rate in noisy environments. Requires continuous monitoring.
Hybrid Analysis	Combines static and behavioral methods. Balances detection speed and accuracy. Effective for complex threats.	More complex to implement. Potentially more resource-intensive. May still miss novel attack vectors.

Security Implications of Linux Namespaces and Cgroups in Containers

Linux namespaces and cgroups are the core technologies that enable containerization by providing process isolation and resource management. Namespaces create isolated environments for each container, ensuring separate views of system resources (e.g., process IDs, network interfaces, file systems) [13]. Cgroups, on the other hand, allocate and limit the resources (e.g., CPU, memory, disk I/O) that containers can consume.

However, these technologies introduce some security challenges due to the shared nature of the host kernel. Containers running on the same host share the kernel, which opens up potential attack vectors. For instance, vulnerabilities in the kernel or misconfigurations in the container runtime can lead to container escapes or privilege escalation attacks. These risks compromise the isolation between containers and the host, unlike traditional virtual machines (VMs), which have more robust isolation due to separate kernels.

This figure 2 could depict a container architecture, highlighting potential attack points, such as the kernel, namespaces, cgroups and inter-container communication, which are susceptible to security vulnerabilities.

Static analysis remained the dominant method for malware detection in containerized environments. However, static techniques faced significant limitations, especially in detecting new and evolving malware. Static analysis usually concentrates on looking for known signatures in container images, but it is unable to detect threats that take advantage of runtime flaws or avoid detection by signature-based tools. Furthermore, it was challenging for static analysis tools to offer reliable coverage because of the dynamic and transient characteristics of containers, which are frequently created and destroyed.

As container technologies continued to mature and threats became more sophisticated, the need for more dynamic and adaptive security methods became evident. Behavioral analysis emerged as a key technique, offering a more flexible and effective means of detecting novel malware in these dynamic environments.

Behavioral Monitoring Techniques in Container Environments

As containerization gained traction in the software development world, security experts had to adapt existing monitoring tools and techniques to ensure the safety of containerized environments [14]. Unlike traditional systems, containers are highly dynamic, ephemeral, and share the same host kernel, presenting unique security challenges. Traditional security methods, such as signature-based malware detection, struggled to stay up with the evolving threat landscape. As a result, behavioral monitoring became crucial in identifying malicious activities in containers. This section explores the key system-level tracing tools, process analysis techniques, and early behavioral monitoring tools used to detect anomalies in containerized environments.

System-Level Tracing: Sysdig, Auditd, and Seccomp Logs

System-level tracing is one of the foundational techniques for monitoring and detecting malicious activities in containerized environments. By capturing and analyzing the system calls made by running processes, these tools provide insights into the interactions between containers and the host system.

Sysdig

sysdig is a powerful tool designed for system visibility and troubleshooting. It can monitor and capture system calls, enabling the detection of potentially malicious activity in containers.

sysdig allows for real-time tracing of system interactions, including the creation of files, network connections, and process creation. This makes it particularly useful for identifying suspicious behavior, such as attempts to exploit kernel vulnerabilities or escalate privileges.

auditd

Another important tool for documenting system activity is the Linux Auditing System, which is represented by auditd. It records a number of activities, such as system calls, file access, and user actions. A useful tool for keeping an eye on container interactions and spotting unusual activity that might point to malware or intrusions is Auditd.

seccomp logs

Seccomp (Secure Computing Mode) is a Linux kernel feature that restricts the set of system calls a containerized process can execute. It helps minimize the attack surface by enforcing syscall whitelists. Monitoring seccomp logs is essential for detecting attempts to invoke blocked or unauthorized syscalls, which may indicate an attempt to bypass security restrictions or exploit kernel-level vulnerabilities. While lightweight and efficient, seccomp's effectiveness depends heavily on how well its syscall filtering policies are defined and enforced.

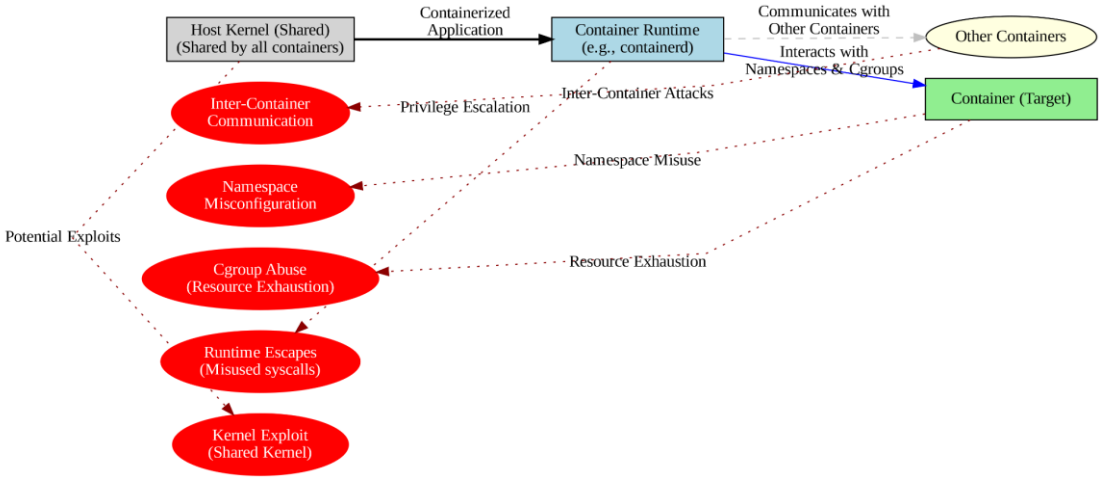


Figure 2: Architecture of Container with Labeled Attack Points

Analyzing Process Tree and Syscall Frequency

A critical aspect of behavioral analysis is examining the way processes interact with each other and the host system. Unusual behaviors such as excessive system calls, abnormal process trees, or strange resource usage can indicate malicious activities [15]. Anomalous behaviors may involve malware executing unusual patterns of system calls or spawning processes in ways that deviate from typical patterns.

Process Tree Analysis

Running processes' hierarchy and interactions are shown by the process tree. A malware infection might show up as a new process or a series of processes straying from the expected process hierarchy. Analyzing these connections helps security experts spot abnormal process execution and find hostile activity including lateral movement between containers or privilege escalation.

Syscall Frequency Analysis

Monitoring the frequency of system calls made by containerized applications can provide another layer of insight. In a normal containerized environment, processes make a predictable

set of system calls at certain frequencies. However, a compromised container might exhibit unusual or frequent system calls that could point to malware attempting to exploit the system or gain unauthorized access.

Early Versions of Behavioral Monitoring Tools: Falco and AppArmor

Several behavioral monitoring tools [16] began to gain popularity within container security:

Falco

Made for containerized systems, Falco is an open-source, runtime security monitoring tool. Using system calls, file accesses, and network activity, it seeks out unusual behavior. Falco can spot possibly harmful behaviors including containers trying to access files or network resources they shouldn't by always tracking container interactions with the host system. Highly flexible, the tool can be set to identify particular harmful trends within the container environment.

AppArmor

While Falco focuses on behavioral detection, AppArmor offers policy-based enforcement. Its security profiles can block containers from performing unauthorized actions, such as accessing restricted directories or modifying kernel parameters. Though early versions lacked dynamic behavioral insights, AppArmor's integration with container runtimes like Docker and Kubernetes adds an important layer of proactive defense by preventing known attack patterns through pre-defined rules.

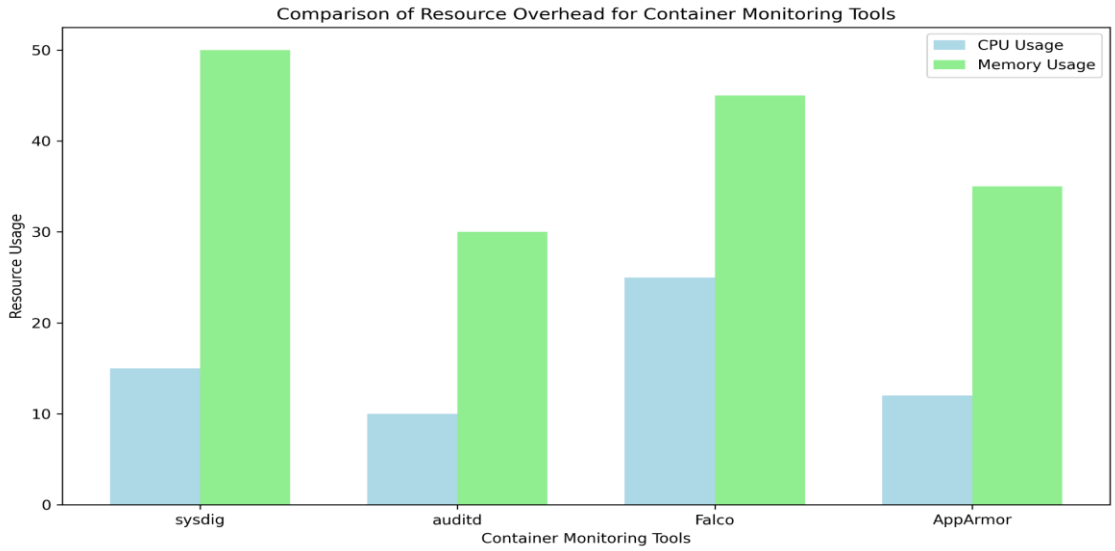


Figure 3: Bar Chart Comparing Overhead of Major Monitoring Tools

This graph would compare the resource overhead (e.g., CPU usage, memory consumption) of popular container monitoring tools such as sysdig, auditd, Falco, and AppArmor. Monitoring the overhead is important because it provides insight into how much resources are consumed by each tool while detecting malware or anomalous behavior. A tool with higher overhead may interfere with the performance of containerized applications, making it less suitable for production environments.

Kernel Instrumentation Techniques: kprobes and eBPF

Kernel instrumentation tools provide deep visibility into the internals of containerized systems, allowing for the detection of malicious activity at the kernel level. These techniques offer

powerful monitoring capabilities but may have certain limitations when it comes to performance and container awareness.

Kprobes

kprobes is a Linux kernel feature that allows dynamic instrumentation of almost any kernel function. It enables developers and security analysts to insert probes at specific points in the kernel and collect information when those functions are executed. In the context of container security, kprobes can be used to trace container-related system calls, monitor file access, or intercept privilege escalation attempts.

Strengths:

i. Flexible placement of probes in arbitrary kernel code.
ii. Minimal performance overhead when used sparingly.
iii. Valuable for debugging and low-level monitoring in research and forensic contexts.

Limitations:

i. Requires kernel-specific knowledge and careful configuration.
ii. Risk of destabilizing the system if improperly implemented.
iii. Not ideal for large-scale, real-time monitoring due to potential performance impact at scale.

eBPF

eBPF extends the capabilities of kprobes by enabling the execution of sandboxed programs in the kernel without modifying kernel code or inserting kernel modules. eBPF can hook into a variety of kernel events, such as system calls, network packets, and tracepoints, making it a flexible and efficient framework for real-time container monitoring.

Strengths:

i. Low-overhead, high-performance monitoring suitable for production environments.
ii. Enables complex logic and filtering within the kernel, reducing user-space processing needs.
iii. Integrates well with modern observability stacks (e.g., Prometheus, Grafana) and security tools (e.g., Cilium, Tracee).

Limitations:

i. Steep learning curve due to kernel constraints and eBPF programming model.
ii. Still evolving, with some features dependent on newer kernel versions.
iii. Security implications of running custom code in the kernel space must be carefully managed.

The Table 3 summarizes the capabilities and limitations of key monitoring tools used in container security. It provides an overview of the tool's granularity of monitoring, support for the Linux kernel, and awareness of container-specific behaviors.

Table 3: Tool Matrix Capabilities, Granularity, Kernel Support, Container Awareness

Tool	Capabilities	Granularity	Kernel Support	Container Awareness
Sysdig	Real-time system calls monitoring, process tracking	Fine-grained (per system call)	Kernel-level support	Full container awareness
Auditd	Event auditing, file system monitoring	Process-level	Kernel-level support	Partial container awareness
Seccomp	Securing system calls, policy enforcement	Fine-grained (per system call)	Kernel-level support	Partial container awareness
Falco	Runtime security, syscall monitoring	Fine-grained (per syscall, file access)	Kernel-level support	Full container awareness
AppArmor	Mandatory access control, policy enforcement	Process-level	Kernel-level support	Full container awareness (with configuration)
Kprobes	Kernel-level function tracing	Very fine-grained	Kernel-level support	Limited container awareness
eBPF	High-performance monitoring, kernel events	Fine-grained (per event)	Kernel-level support	Limited container awareness

The behavioral monitoring techniques described in this section provide a foundation for understanding the evolution of container security. By shifting from static analysis to more dynamic, real-time monitoring tools, the security community has made significant strides in detecting and mitigating threats within containerized environments. These techniques are critical in ensuring that containers remain secure as they continue to play a central role in modern application deployment.

Fine-Grained Behavioral Features Used in Detection

As malware techniques became increasingly evasive and polymorphic in containerized environments, the need for fine-grained behavioral detection mechanisms grew significantly [17]. Traditional binary-based or static approaches were limited in their ability to detect novel malware variants, especially in ephemeral and resource-constrained container environments. Researchers and security systems started to create detection models able to detect subtle, aberrant behaviors suggestive of compromise by concentrating on exact behavioral indicators down to system calls, memory use, and access patterns. Key behavioral traits applied and their significance in container malware detection are described in this part.

Syscall Sequences (n-gram Modeling & LSTM Approaches)

System calls sequences serve as a low-level yet powerful feature for behavior modeling. Since all high-level actions by applications ultimately translate to sequences of syscalls, they provide a reliable lens through which malicious activity can be observed.

- Among the first methods to describe normal and anomalous syscall behavior were n-gram models, which record fixed-length sequences of syscalls—e.g., trigrams or 5-grams. Although basic, these statistical models were good in spotting deviations from accepted standards of performance.
- LSTM (Long Short-Term Memory) models, a type of recurrent neural network (RNN), began gaining traction before 2021 for their ability to model longer temporal dependencies in syscall patterns. LSTMs showed promise in distinguishing between benign and malicious container behavior without requiring prior knowledge of specific malware signatures.

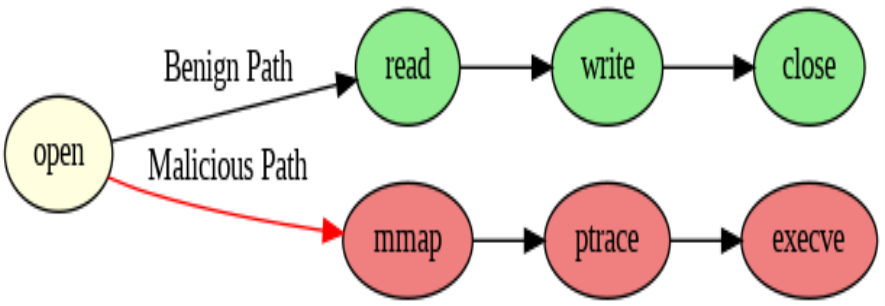


Figure 4: Example syscall trace showing benign vs. malicious pattern

This figure would visualize syscall sequences during normal operation (e.g., standard input/output, file read/write) compared with a trace collected during a known attack (e.g., privilege escalation, kernel exploit), highlighting the divergence in call patterns.

File Access Patterns & Entropy Detection

Malware often interacts with files in unusual ways scanning directories, modifying configurations, or writing encrypted payloads. Behavioral monitoring of file access can reveal several red flags:

- Frequent access to sensitive files (e.g., /etc/passwd, /var/log)
- High-entropy writes, which might be a sign of payload obfuscation or cryptographic techniques used in ransomware
- Unusual read/write patterns such as recursive file access or unexpected access timestamps

These indicators were commonly used to flag unauthorized activity in container instances where ephemeral storage might otherwise make forensics difficult.

Network Socket Creation & DNS Call Frequency

Another crucial component of behavior-based detection is network behavior monitoring. Numerous malware variations try lateral movement, exfiltrate data via DNS, or establish outgoing connections to command-and-control (C2) servers.

- Excessive or rapid DNS resolution attempts can be an indicator of domain generation algorithms (DGAs) commonly used by malware.

- Backdoors or reverse shell activity may be indicated by repeated or suspicious socket creations, particularly to unusual ports or public IPs.
- Correlating network activity with process and user identity adds context that improves detection fidelity in multi-tenant container clusters.

Memory Usage Spikes & Fork-Bomb Heuristics

Containers are typically subject to strict resource quotas. A sudden spike in memory usage or an unexpected surge in child processes (e.g., via fork ()) can indicate malicious behavior such as:

- Fork bombs, designed to overwhelm a host by recursively spawning processes
- Techniques for memory spraying that are used in exploits
- In-memory payload injection that circumvents disk-based detection

Behavioral baselining of resource use was a common pre-2021 approach, often integrated into container orchestrators or sidecar monitoring agents.

Process Injection & Privilege Escalation Indicators

Advanced threats in containers sometimes attempt process injection (e.g., ptrace hijacking) or escalation to root privileges, particularly in weakly isolated environments.

- Behavior-based signals included the use of ptrace, execve chains, or modifications to /proc entries
- Lateral movement or breakout attempts were frequently preceded by abnormal parent-child process relationships (e.g., nginx spawning a shell).
- Escalation attempts were correlated with changes in user ID (setuid(), setgid()) or manipulation of container runtime flags

These features, while subtle, became essential in early behavior-based intrusion detection systems (IDS) such as Falco and in customized eBPF security monitors.

Detection Algorithms and Models Used

As containerized environments gained popularity in the late 2010s, several machine learning–based detection models were adapted and evaluated for malware detection [18]. These algorithms many of which had been successful in earlier host-based or virtualized setups were repurposed to handle container-specific behavioral telemetry such as syscall traces, network activity, and resource usage patterns. However, the algorithms and models employed were largely limited by their training data volume, architectural constraints, and inability to generalize to the rapidly evolving threat landscape. This section highlights the core techniques that dominated behavior-based detection, which have since been surpassed by more scalable, container-native solutions.

Decision Trees, Random Forests, and SVMs: Non–Deep Learning ML Models

Traditional supervised learning models such as Decision Trees, Random Forests, and Support Vector Machines (SVMs) were among the earliest ML techniques applied to system behavior classification tasks [19].

- Because of their interpretability and low processing demands, decision trees and random forests were preferred. Syscall frequency patterns or categorical features (like file write flags or network port access) could be mapped to benign or malicious labels using these models.

- SVMs were applied for their ability to handle high-dimensional syscall vectors, often using radial basis function (RBF) kernels.

Despite their usefulness in detecting known behavioral signatures, these models struggled with generalization to new malware families and often required extensive feature engineering.

Hidden Markov Models (HMMs) and Clustering via K-Means

To model the sequential nature of syscalls and process behaviors, researchers experimented with probabilistic sequence models and unsupervised clustering techniques [20].

- Hidden Markov Models (HMMs) were employed to represent typical syscall transitions in benign applications and flag sequences with low likelihood scores.
- K-Means clustering was used to group similar behavioral traces, with anomalies identified as low-density or distant samples from the main clusters.

These models, while innovative at the time, lacked the capacity to scale to highly dynamic container workloads and were prone to high false positives in multi-tenant clusters.

Early LSTM-Based Detection

An important step toward deep learning-driven detection was the development of LSTM (Long Short-Term Memory) networks. LSTMs were particularly well-suited to modeling syscall streams and multivariate time-series data because they could learn temporal dependencies in system behavior.

However, implementations had notable constraints:

- Training datasets were often small and lacked diversity across container types or workloads.
- Models were typically trained offline and struggled to adapt in real-time container orchestrations.
- Interpretability remained an issue, limiting operational adoption in SOC (Security Operations Center) pipelines.

Still, LSTMs set the stage for future work in neural sequence modeling for malware detection.

Signature-Augmented Behavioral Detection (Hybrid Approaches)

Several systems relied on hybrid detection models that combined behavioral monitoring with signature-based lookups:

- Signature engines were used to detect known malware families (e.g., via rule-based systems like YARA or ClamAV).
- Static rules were combined with ML model anomaly scores (such as unexpected port access or syscall outliers) to generate composite alerts.

While these hybrids improved detection rates for known threats, they were ultimately limited by signature drift and inability to identify zero-day threats or behavioral mimicry.

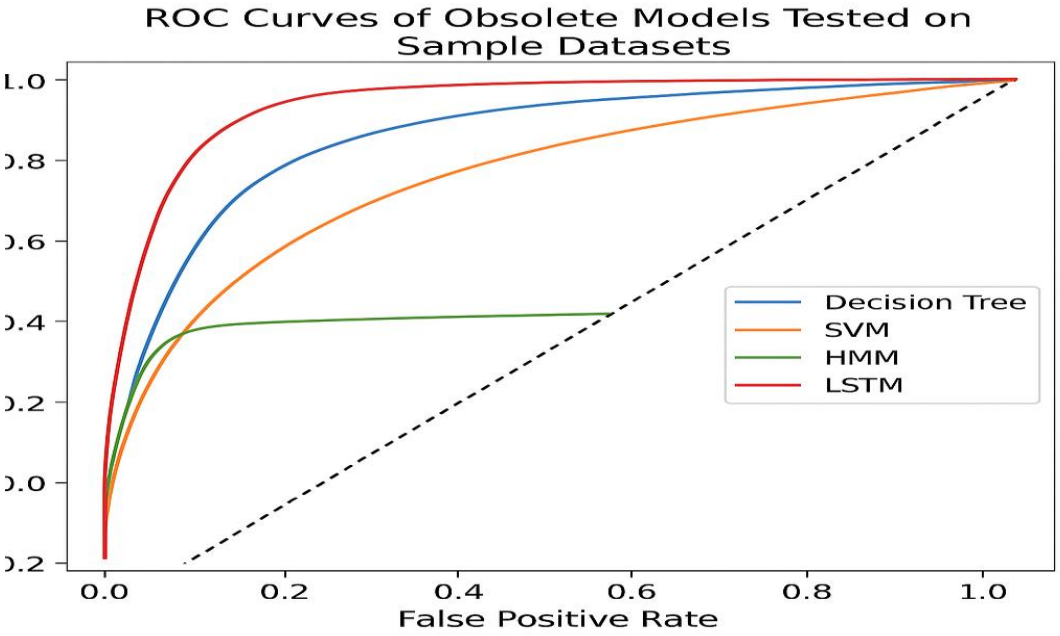


Figure 5: ROC Curves of Obsolete Models Tested on Sample Datasets

The figure 5 could show comparative Receiver Operating Characteristic (ROC) curves of models like SVM, Random Forest, LSTM, and HMM on benchmark behavioral datasets. It would illustrate trade-offs between true positive rates and false positives, with older models showing weaker AUC (Area Under Curve) scores.

Table: Summary of ML Models Used for Container Malware Detection

Model Type	Approach	Detection Strengths	Limitations
Decision Trees / Random Forests	Supervised, interpretable	Fast, transparent logic trees	Required manual features, overfitting on small sets
SVM (RBF Kernel)	Supervised, nonlinear	Good for binary syscall classification	Limited scalability, hard to tune
HMM	Probabilistic sequence	Modeled syscall transitions well	Poor generalization, sensitive to noise
K-Means Clustering	Unsupervised	Helped flag outliers in syscall patterns	High false positive rate, no time-series modeling
LSTM (Early Use)	Neural sequence modeling	Captured long-range syscall dependencies	Small training sets, black-box nature
Hybrid (Signature + Behavior)	Rule-based + ML	Effective for known malware + anomalies	Poor zero-day coverage, signature maintenance

Early models laid the foundation for container-native security, but their static assumptions and limited context-awareness underscored the need for more scalable, adaptive approaches. This gap has been addressed with technologies like eBPF, richer telemetry pipelines, and real-time behavioral baselining. Modern deep learning methods—particularly CNNs and GNNs—offer improved detection by leveraging complex container data. CNNs enable pattern recognition from syscall and network activity streams, while GNNs model relationships between processes, containers, and orchestration metadata, enabling context-aware and cross-container threat detection. These models are better suited for identifying sophisticated behaviors missed by earlier techniques.

Limitations of These Techniques in Modern Context

As the container ecosystem matured particularly with the rise of Kubernetes-based orchestration and microservices architecture the detection techniques and models used began to show clear limitations. While they provided early promise for identifying behavioral anomalies in container workloads, they could not scale effectively or adapt to the increasing complexity of cloud-native environments. This section examines the critical shortcomings of these legacy approaches in the context of modern infrastructure demands.

Inefficiency in Cloud-Native & Kubernetes-Based Orchestration

Early detection tools were not designed with cloud-native principles in mind. Many solutions assumed static container lifecycles, limited horizontal scaling, and minimal orchestration dynamics [21].

- These tools lacked awareness of transient container lifespans, dynamic scaling, and frequent image redeployments common in Kubernetes.
- Many models required full logs, complete traces, or centralized processing—unsuitable for ephemeral, distributed, and large-scale container environments.

High False Positive Rates

Behavioral models from this era especially those relying on simple frequency or clustering methods often produced high false positive rates when exposed to real-world workloads.

- Legitimate but rare workload behaviors (e.g., backup scripts, ephemeral processes) were misclassified as malicious.
- The models produced noisy alerts because they lacked contextual awareness (such as time-of-day patterns or container role) and adaptive baselining.
- Security teams faced alert fatigue and low signal-to-noise ratios, reducing trust in the systems.

Lack of Cross-Container Correlation or Awareness

Most legacy behavioral tools focused on analyzing individual containers in isolation.

- They could not identify attack patterns that spanned multiple containers or pods, such as lateral movement or shared volume exploits.
- There was no ability to track identity, context, or behavioral continuity across container restarts, replicas, or service meshes.
- Without cross-container telemetry stitching, sophisticated threat campaigns evaded detection by staying below per-container anomaly thresholds.

Inability to Detect Sophisticated APTs or Polymorphic Malware

Advanced Persistent Threats (APTs), polymorphic malware, and living-off-the-land techniques require deeper behavioral inference and context-aware analytics.

- Legacy tools lacked memory introspection, process lineage analysis, or in-depth runtime correlation to detect stealthy persistence techniques.
- Polymorphic code frequently used in evasive container malware escaped signature-augmented models entirely.
- These tools failed to account for malware that adapts behavior based on environmental signals (e.g., sandbox detection evasion)

No Support for Runtime Context-Awareness

Modern container security requires embedding detection within runtime context accounting for process ancestry, runtime flags, container metadata, and pod-level configurations.

- Models ignored key runtime parameters such as container labels, namespaces, cgroups hierarchy, and deployment intent.
- Lack of integration with orchestration APIs (Kubernetes RBAC, service accounts, network policies) meant that behavioral anomalies were assessed without understanding user roles or policy violations.
- As a result, decisions were made in a vacuum, leading to poor detection fidelity.

Ethical and Legal Concerns in Multi-Tenant Environments

Behavioral telemetry collection in containerized infrastructures especially within shared or multi-tenant clusters raises ethical and legal considerations.

- Monitoring tools that capture detailed system calls, file access patterns, or process lineage may unintentionally expose sensitive data or user-specific behaviors, especially in SaaS or cloud-hosted environments.
- Compliance with data privacy regulations such as GDPR, HIPAA, or CCPA becomes challenging when behavioral logging is insufficiently scoped or lacks clear tenant separation.
- Researchers and practitioners must weigh detection accuracy against user privacy, ensuring that telemetry collection aligns with least privilege principles and includes opt-in policies or pseudonymization techniques where required.

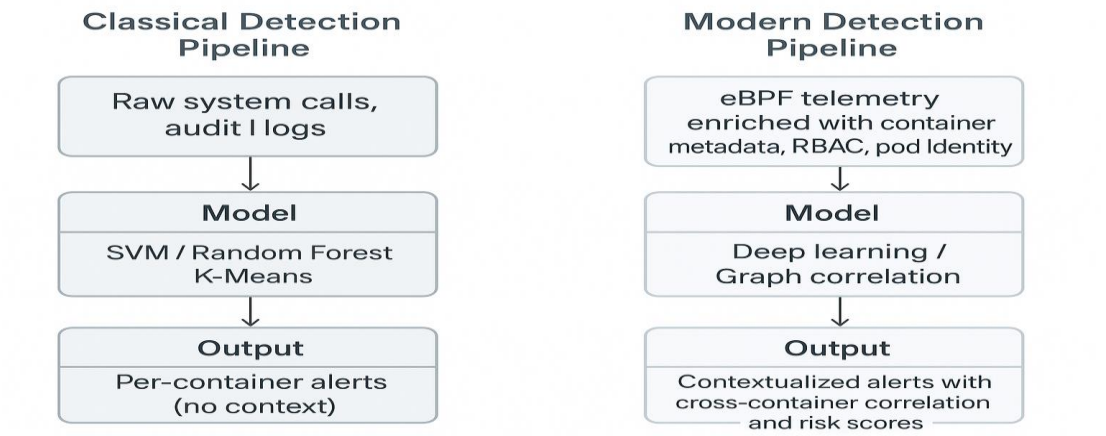


Figure 6: Transition from Traditional to Modern Behavioral Detection Architectures

This figure 6 contrasts outdated behavioral detection methods with modern, container-native approaches. It emphasizes how contemporary architectures address the limitations of legacy tools by enabling real-time, scalable, and context-rich threat detection across dynamic containerized workloads.

CONCLUSION AND RECOMMENDATION

Conclusion

This research has explored the landscape of fine-grained behavioral malware detection techniques as applied to containerized environments. At the time, detection strategies centered on low-level system telemetry such as syscall sequences, process trees, and file access behavior paired with lightweight machine learning models. By providing more flexibility to new threats within container runtimes, these tools and approaches closed a significant gap left by conventional signature-based detection

Despite their innovation at the time, the approaches examined in this study demonstrated several limitations when viewed through the lens of contemporary cloud-native infrastructure. Techniques based on static syscall modeling or isolated container analysis struggled with scalability and precision. They frequently produced high false positives, lacked context-awareness (e.g., orchestrator metadata or cross-container relationships), and failed to detect stealthy or polymorphic malware such as Advanced Persistent Threats (APTs). Although tools like Sysdig and Auditd provided useful observability, their applicability in production workloads was limited by their quantifiable performance overhead.

Nonetheless, these early techniques laid a vital foundation. They contributed to a broader shift from signature-based toward behavior-based security models and fostered the development of real-time, fine-grained monitoring paradigms that informed the next generation of tools. The

historical techniques covered in this research now serve as an important benchmark against which modern solutions can be measured.

Recommendations

Looking forward, future research should focus on integrating high-fidelity behavioral signals with context-rich runtime environments, leveraging technologies like extended Berkeley Packet Filter (eBPF), Graph Neural Networks (GNNs), and orchestrator-level visibility. While these advancements are outside the scope of this paper, they directly address the drawbacks and achievements of the methods discussed here. Thus, understanding these outdated methods is not merely of archival value it is critical to shaping the trajectory of container security in the years ahead.

References

[1] J. Watada, et al., "Emerging trends, techniques and open issues of containerization: A review," IEEE Access, vol. 7, pp. 152443–152472, 2019

[2] Y. Vlasov, N. Khrystenko, and D. Uzun, "Analysis of Modern Continuous Integration/Deployment Workflows Based on Virtualization Tools and Containerization Techniques," in Integrated Computer Technologies in Mechanical Engineering: Synergetic Engineering, Cham: Springer International Publishing, 2020.

[3] T. Siddiqui, S. A. Siddiqui, and N. A. Khan, "Comprehensive analysis of container technology," in Proc. 2019 4th Int. Conf. Information Systems and Computer Networks (ISCON), Mathura, India, 2019.

[4] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," IEEE Access, vol. 7, pp. 52976–52996, 2019.

[5] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, "Behavior-based features model for malware detection," J. Comput. Virol. Hacking Tech., vol. 12, pp. 59–67, 2016.

[6] A. Gómez Ramírez, Deep learning and isolation-based security for intrusion detection and prevention in grid computing, Ph.D. dissertation, Frankfurt U., 2018.

[7] A. Samir, et al., "Anomaly detection and analysis for reliability management clustered container architectures," Int. J. Adv. Syst. Meas., vol. 12, no. 3, pp. 247–264, 2020.

[8] A. Khan, "Key characteristics of a container orchestration platform to enable a modern application," IEEE Cloud Comput., vol. 4, no. 5, pp. 42–48, 2017.

[9] M. Pearce, S. Zeadally, and R. Hunt, "Virtualization: Issues, security threats, and solutions," ACM Comput. Surv., vol. 45, no. 2, pp. 1–39, 2013.

[10] G. Suarez-Tangil, et al., "Evolution, detection and analysis of malware for smart devices," IEEE Commun. Surveys Tuts., vol. 16, no. 2, pp. 961–987, 2013.

[11] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," IEEE Access, vol. 8, pp. 6249–6271, 2020.

[12] S. Talukder, "Tools and techniques for malware detection and analysis," arXiv preprint arXiv:2002.06819, 2020.

[13] S. M. Jain, Linux Containers and Virtualization: A Kernel Perspective, 2020.

[14] A. Simioni, "Implementation and evaluation of a container-based software architecture," M.S. thesis, 2017.

[15] S. M. Varghese and K. P. Jacob, "Process profiling using frequencies of system calls," in Proc. 2nd Int. Conf. Availability, Reliability and Security (ARES), Vienna, Austria, 2007.

[16] H. Gantikow, et al., "Rule-based security monitoring of containerized environments," in Proc. Int. Conf. Cloud Computing and Services Science, Cham: Springer International Publishing, 2019.

[17] O. Or-Meir, et al., "Dynamic malware analysis in the modern era—A state of the art survey," ACM Comput. Surv., vol. 52, no. 5, pp. 1–48, 2019.

[18] F. Liang, et al., "Machine learning for security and the internet of things: the good, the bad, and the ugly," IEEE Access, vol. 7, pp. 158126–158147, 2019

[19] V. Rodriguez-Galiano, et al., "Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines," Ore Geol. Rev., vol. 71, pp. 804–818, 2015.

[20] N. Pant and R. Elmasri, "Detecting meaningful places and predicting locations using varied k-means and hidden Markov model," in Proc. 17th SIAM Int. Conf. Data Mining (SDM), 3rd Int. Workshop on ML Methods for Recommender Systems, Houston, TX, USA, 2017

[21] Z. Zhong and R. Buyya, "A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources," ACM Trans. Internet Technol. (TOIT), vol. 20, no. 2, pp. 1–24, 2020.

License