Enhancing Container Runtime Security: A Case Study in Threat Detection

Amina Eldjou[1,2,*], Moahmed Elhadi Amoura[1], Mohcene Soltane[1], Meriem Belguidoum[1,3], Samir Bennacer[4] and Ilham Kitouni[1,2]

[1]*University of Constantine 2 Abdelhamid Mehri Constantine, Algeria*
[2]*LISIA Laboratory University Abdelhamid Mehri*
[3]*LIRE Laboratory University Abdelhamid Mehri*
[4]*Octodet, London, England*

Abstract
In recent years, advances in software development, operation, and maintenance technologies have driven the increasing prevalence of Cloud native architecture. This architectural approach offers efficient virtualization, resource isolation, and effective management capabilities. However, ensuring robust security within these dynamic and scalable environments is crucial. Traditional security tools suffer from a lack of visibility and effectiveness in threat detection. Research studies have suggested log management solutions, log aggregation, and real-time log analysis to identify patterns, anomalies, and threat detection. While these solutions are effective, they suffer from a lack of visibility, which is our research focus. This paper suggests a comprehensive solution that aims to enhance threat detection in containerized systems. By leveraging the combined capabilities of Cilium Tetragon and Elastic Stack, the proposed solution offers an easy integration that enhances the visibility of containerized environments and Simplifying the process of analyzing logs. This integration empowers security professionals to gain comprehensive insights, enabling them to efficiently detect and address any anomalous or malicious activities occurring within the container runtime.

Keywords
Cloud native, Threat detection, Container runtime, Cybersecurity

1. Introduction

Over the past few years, the increasing advancements in software development, operation, and maintenance technologies have led to the increased prominence of Cloud native architecture. This architectural approach has gained popularity due to its distinctive characteristics, such as efficient virtualization, the ability to isolate resources, and effective management capabilities. 37% of organizations have experienced a container security incident, this number is expected to increase in 2023, as more and more organizations adopt containerized applications.

Cloud native is a term used to describe a set of technologies and practices that are designed to build and run scalable applications in modern, dynamic environments such as Cloud computing

TACC 2023: Tunisian-Algerian Joint Conference on Applied Computing, November 06–08, 2023, Sousse, Tunisia
*Corresponding author.
amina.eldjou@univ-constantine2.dz (A. Eldjou); mohamedelhadi.amoura@univ-constantine2.dz (M. E. Amoura); mohcene.soltane@univ-constantine2.dz (M. Soltane); meriem.belguidoum@univ-constantine2.dz (M. Belguidoum); samir@octodet.com (S. Bennacer); ilham.kitouni@univ-constantine2.dz (I. Kitouni)

platforms [1].

The term Cloud native was coined by the Cloud native Computing Foundation (CNCF) [1]. The CNCF defines Cloud native as technologies that are built to run in a Cloud environment and benefit from its elasticity, distributed architecture, and ability to rapidly release new features. However, ensuring robust security within these dynamic and scalable ecosystems is crucial. Traditional security tools face challenges in containerized environments due to the nature of container infrastructures [2]. According to the Cloud native Computing Foundation, container usage in production has seen a great increase between 2016 and 2022[3]. In the Cloud native computing and containers world, security plays a crucial role like any other platform or system. While the recent State of Kubernetes 2022 survey highlighted the continual rise of Cloud native adoption among organizations, it has also become a popular target for threats and vulnerabilities [4]. By implementing effective security measures, organizations can mitigate risks, protect sensitive data, and provide a detailed investigation report. For example, a Kubernetes pod may run for a short time before being automatically terminated and its resources reused [5]. Traditional security tools may not be able to detect suspicious activity in a timely manner, and they may not be able to collect the necessary data to investigate an attack. Additionally, some traditional security tools, such as web application firewalls, next-generation firewalls, and endpoint security, are not effective in container environments because they do not have visibility into the entire container ecosystem.

Taking this further, the current study focuses on a comprehensive approach to enhance threat detection using Cilium Tetragon and Elastic Stack. This integration offers deep visibility into containerized environments, simplifying log analysis for professionals to identify abnormal or malicious activities in real time. The solution addresses timely threat detection by enabling real-time monitoring and alerting. It empowers security operation center (SOC) analysts to promptly respond to potential threats in the container runtime environment. In summary, this paper proposes a solution to enhance threat detection in containerized systems through powerful visibility tools and real-time log analysis, aiding SOC analysts in identifying threats or malicious behavior. The paper's structure includes Section 2 for foundational insights into Cloud native computing and security considerations, Section 3 exploring relevant threat detection tools, Section 4 detailing the proposed approach, Section 6 interpreting findings and suggesting future directions, and Section 7 concluding the paper and outlining future work.

2. Background

2.1. Cloud native

Cloud native computing is an approach to developing and deploying applications that takes full advantage of Cloud computing principles [1]. A new approach to software development emerged as a result of the need for more efficient, scalable, and agile software solutions in modern IT environments. Cloud native applications are software programs that consist of multiple small, interdependent services called microservices. The key characteristics of Cloud native applications include firstly, Microservices architecture decomposes applications into small, loosely coupled services by functionality area [1], offering benefits like increased agility, scalability, and maintainability [6]. Secondly, the Service Mesh is a dedicated infrastructure layer that simplifies communication between microservices in Cloud-native applications [1], enhancing development, deployment, and management [1, 6]. Containerization encapsulates software with OS components into portable containers, with Docker containers as a common example [7, 8, 9]. Linux OS utilizes namespaces and control groups to achieve isolation and resource management [1, 10]. Container orchestration platforms automate deployment and scaling [11, 1], while Continuous Integration and Continuous Delivery (CI/CD) streamline code development and deployment [12, 1]. Lastly, Serverless, a Cloud-native model [1], enables server-free application development [13]. These concepts collectively shape the Cloud-native ecosystem, as supported by referenced literature.

2.2. The 4C's of Cloud Native Security

Ensuring robust security in Cloud native environments involves a layered security approach. The four C's represent the essential layers of Cloud native security: Cloud, Code, Container, and Cluster. Developers and DevOps teams must adhere to Cloud computing best practices across these layers to achieve security goals before user access. Each layer builds upon the next, and addressing security at the Code level relies on the foundation provided by the Cloud, Cluster, and Container layers [14, 1, 4].

Code, at the core of applications, demands robust security measures like access management, threat monitoring, and TLS encryption to mitigate risks [14]. Once developed, code can be containerized, packaging it and its dependencies into lightweight, portable units [1]. This containerization ensures consistent, efficient deployment across diverse environments, maintaining the application's reliability on various systems [3]. Clusters, comprised of individual services or workloads running in separate containers, interconnected over a network, form an integral part of this ecosystem [5]. Cloud computing services provide on-demand IT resources over the internet with pay-as-you-go pricing, operating within virtualized computing environments and enabling easy access to a multitude of services and applications without the need for physical hardware [15].

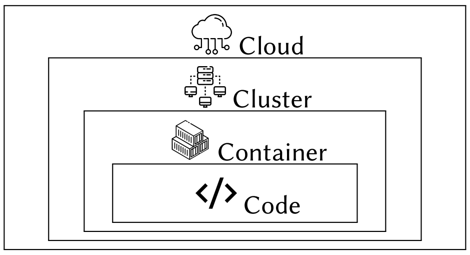

Table 1
The 4C's of Cloud Native Security

Thus, each layer of the Cloud Native security model is interdependent. A code layer's strength is reinforced by its underlying security layers (Cloud, Cluster, Container). This layered approach augments the defense in depth computing approach to security, which is widely regarded as a best practice for securing software systems [14]. This study will focus on the container layer specifically the runtime security.

2.3. Container runtime security

Container runtime security is critical in the cloud computing landscape, especially in microservice architectures where containers are prevalent for efficient, scalable, and portable application deployment [1, 16, 17]. However, the use of containers introduces new vulnerabilities and attack vectors, posing risks to both applications and host systems' security and integrity [10]. To address these challenges comprehensively, a holistic approach to container runtime security is essential [10].

During an application's active runtime, unforeseen security risks may emerge, including overlooked vulnerabilities or setup errors from the build phase [17]. Attackers can exploit these weaknesses, necessitating real-time monitoring for unusual behavior. Anomaly detection at runtime can identify privilege escalations, cryptomining, unexpected network flows, container escape attempts, and other insecure activities.

Linux kernel features play a pivotal role in container security, leveraging namespaces, control groups (cgroups), and Seccomp [17, 10, 18, 19]. Namespaces segregate system resources, ensuring container independence [18, 19]. Cgroups allocate and restrict resources, preventing contention and ensuring fair utilization [18, 19]. Seccomp filters system calls, reducing the attack surface [18, 19].

Within this context, the Berkeley Packet Filter (BPF) and its extension, eBPF, enable advanced observability and tracing in container environments, enhancing security [18, 19, 10]. BPF safely executes programs triggered by kernel events, offering safety assurances against crashes and malicious actions [18, 19]. It provides real-time security observability, speed, and convenience for monitoring high-volume event data, making it a safer and efficient option compared to traditional methods [20, 21]. By offering deeper insights into observability, eBPF ensures secure, non-intrusive telemetry data collection from the entire system [21].

2.4. Threat detection in Cloud native

. Threat detection is the identification, location, and reporting of activities or weapons that could harm a system, network, or target. It can be real-time or retroactive, automated or manual, distinct from threat response, which aims to counter threats. In the dynamic cybersecurity field, effective threat detection is crucial for proactively addressing vulnerabilities and preventing cyber-attacks, data breaches, and security compromises. Recognizing unauthorized activities and anomalies in system behavior is key to mitigating potential damage inflicted by cyber adversaries [22].

To achieve this, there exist two prominent methodologies for threat detection [22]: First, the system makes a detailed profile of the normal activities of the system, network, or program to detect anomalies. Malicious behavior is defined as any deviation from the baseline. Second, signature-based detection is used to detect previously known malicious activities that match a signature or rules-based protocol that model the user's behavior.

Through the systematic analysis of system behavior and the continuous monitoring, threat detection stands as a proactive force that contributes significantly to safeguarding digital assets and maintaining the integrity of critical systems and networks [16]. In the context of container runtime environment threat detection requires a multi-faceted approach that combines anomaly-based detection, real-time monitoring, intrusion detection systems, and the integration of threat intelligence. By continuously monitoring container behavior, network traffic, and system interactions [17], organizations can identify and respond to threats in a timely manner, enhancing the security posture of their containerized applications and Cloud native infrastructures.

Current solutions can be used to enhance container security. The use cases are [10]: (I) protecting a container from applications inside it, (II) inter-container protection, (III) protecting from containers, and (IV) protecting containers from a malicious host. Available solutions for the four use cases can be either(i) software solutions such as Linux namespaces, CGroups, capabilities, seccomp, and LSMs, or (ii) hardware solutions such as using vTPMs and utilizing trusted platform support.

Many researchers have similarly suggested that more studies are required to standardize the processes for deploying containers, defining communication protocols, and establishing assessment techniques.[17, 10].Standards and frameworks should consider various factors such as platform-specific needs, application requirements, practical implementation, automation capabilities, simplicity, efficiency, and ease of integration.

3. Related work

This section reviews and explores some relevant tools for threat detection in the context of container runtime, as they share some similarities and objectives, although there are many commercial solutions available in the same area that are not covered in this paper. In recent years, containerization has revolutionized the deployment and management of applications, but it has also introduced security challenges [10]. These include potential isolation breaches leading to privilege escalation, complexities in orchestrating secure configurations, and attacking private registry [10, 23]. To tackle container runtime security and threat detection within containerized environments, several techniques have been proposed, Container security encompasses several strategies and techniques. Dynamic Analysis Techniques (DAT) involve runtime monitoring of system activities, network traffic, and application behavior to identify anomalies and potential threats [24, 25]. Container Orchestration and Management (COM), using platforms like Kubernetes, provide built-in monitoring and centralized visibility into containers' state, health, resource usage, and network interactions [3]. Intrusion Detection Systems (IDS) monitor network traffic and system activities for malicious or unauthorized behavior, whether at the network level (NIDS) or within containers and hosts (HIDS) [26, 27, 28]. Machine Learning for Threat Detection (MLTD) employs AI techniques to analyze data, detecting patterns and anomalies indicative of threats [29, 30]. Continuous Security Monitoring (CSM) integrates security throughout the container lifecycle, ensuring compliance from development to operation [17]. Finally, Security by Design (SBD) incorporates runtime threat detection into the design phase, enabling early vulnerability identification and threat mitigation [31]. These approaches collectively fortify container security, as supported by the cited references. The container runtime security landscape has witnessed significant innovation through the utilization of eBPF (Extended Berkeley Packet Filter) technology. Falco is an open-source threat detection engine that monitors the Linux kernel to detect anomalies in Kubernetes nodes and containers. Cilium Tetragon enhances container security with fine-grained network policies, while Elastic's CWP 2023 aims to protect Kubernetes containers holistically. Academic research by EL Khairi et al. focuses on system calls in containers to identify deviations that could signal malicious activities. While these solutions offer valuable approaches to container security, they face challenges such as performance overhead, complexity in policy management, and potential false positives and negatives. These limitations highlight the need for better visibility into containerized application behavior While the previous solutions offer innovative approaches to container security, they are not without their challenges and limitations. Real-time monitoring solutions like Falco, which operate at the kernel level, can introduce performance overhead, affecting the throughput and efficiency of the containers being monitored. With its focus on fine-grained network policies, Cilium Tetragon adds an element of complexity that could be challenging for teams without specialized knowledge. Furthermore, these tools often grapple with the issue of false positives and negatives, which could either raise unnecessary alarms or fail to detect subtle, sophisticated attacks. Elastic's CWP 2023, although Cloud native, may face similar performance and accuracy trade-offs, while academic research like that of EL Khairi et al. often necessitates validation in real-world scenarios to ascertain its practical applicability. Additionally, Tetragon's lack of alert generation may hinder threat identification. Complexity in policy configuration and enforcement can delay security deployment, and security tools may not cover all attack vectors [32]. Addressing these limitations is crucial for a comprehensive security framework in the containerized application landscape. The proposed solution combines Cilium Tetragon and Elastic Stack for real-time threat detection, aiding SOC Analysts in identifying and alerting to malicious behavior in container runtimes [32, 33].

4. Methodology

This section aims to present the solution schema proposed. Figure 1 demonstrates how relevant data are collected from various sources to gain visibility into the investigated environment, including network traffic, filesystem events, and process execution using Cilium Tetragon. The data collection process is well defined to ensure a comprehensive scope; this means that all relevant data points and variables that are needed to answer research questions and data integrity ensure that the collected data are trustworthy, reliable, and suitable for analysis. Then the collected data are stored in Elasticsearch; a suitable platform capable of handling large amounts of data and providing efficient querying capabilities and detection rules. The major step before creating the detection rules is performing common transformations on the data before indexing following the elastic common scheme ECS to normalize the event data, which can better analyze, visualize and correlate the data represented in the events.

Due to the conditions and challenges in the container environment such as dynamic scaling, resource management, and network architecture. These are indeed critical considerations in the field of containerization and are recognized challenges in managing and securing containerized applications. Cilium Tetragon, a key component of the Cilium project focusing on enhancing network security in container environments, adeptly addresses various container conditions, including the intricacies of dynamic scaling. Tetragon excels in resource management through automatic scaling, adapting seamlessly to changes in container count within Kubernetes clusters.

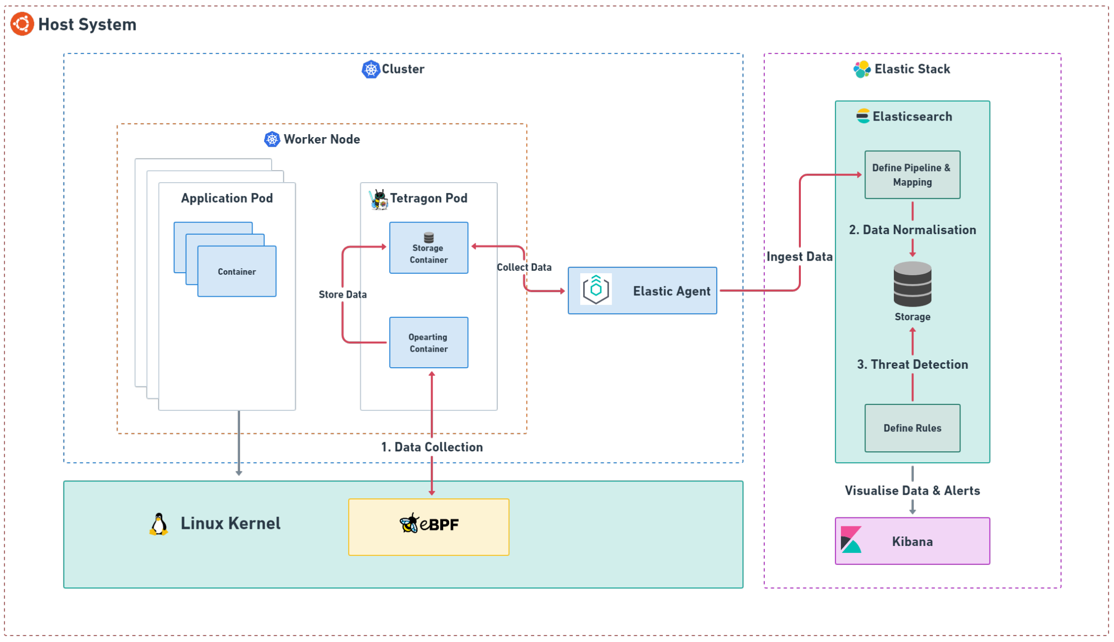

Figure 1: Solution Roadmap.

It ensures real-time visibility and monitoring, maintaining uninterrupted insights into network traffic and security events during dynamic scaling. Elasticsearch integration enables efficient data storage and retrieval. Tetragon enforces dynamic security policies, even as containers scale, and leverages eBPF technology for performance efficiency. Seamlessly integrated with Kubernetes, it offers continuous visibility and adaptive resource management.

The roadmap is divided into three phases:

- **Data collection**: This phase involves collecting data from various sources, such as container logs, network traffic, and filesystem activity.
- **Data normalization**: This phase involves transforming the collected data into a format that can be easily analyzed.
- **Threat detection**: This phase involves using techniques to identify patterns of behavior that are indicative of malicious activity. Figure 1 illustrates the solution roadmap

4.1. Data Collection

Collecting data from the 4C's in a Cloud native application is essential for monitoring and understanding the security posture of the application, by collecting data from these 4C's, organizations can gain a better understanding of their Cloud native application. In this phase, relevant data will be collected from Cloud native applications, including network traffic, software and infrastructure logs, traces, and metrics from the environment and other relevant information. In this phase, The system events are generated by a host kernel, by deploying a test application along with Tetragon in a Kubernetes cluster which is a Cloud native application. This will allow Tetragon to collect the system and container events running in the cluster, such as file access, network activity, system calls, and process execution. Tetragon also performs intelligent filtering and aggregation of events directly in the kernel, reducing the overhead and improving the efficiency of data collection. Furthermore, Tetragon supports real-time runtime enforcement of security policies across the operating system, preventing malicious or unwanted behaviors during container runtime run by the application itself or users with Kubernetes access permission.

A requirement for Tetragon's functionalities is that the system host kernel must support eBPF technologies. The collected logs are saved under Tetragon's specific container, and the next step is to use it as a dataset for future analysis and visualizations to perform the Threat Detection Solution using search and analysis platforms such as the Elastic Stack for further treatment and normalization.

4.2. Data Normalisation

The normalization process consists of several procedures, such as Stored Data Recovery, Data Indexing, and Visualization for Analysis. Each procedure employs its appropriate solution and tools (Elastic stack in this case) to complete the normalization phase.

Stored Data Recovery: The data collection produced log files that were stored within Tetragon containers in the Kubernetes environment. A bridge will be established between the Kubernetes environment and the host system using Elastic Stack solutions such as Elastic Agent. This agent will hook inside the Kubernetes node where the log files are saved during the containers' runtime and communicate with Elasticsearch to create and store a copy of these logs for further procedures using an elastic agent.

Indexing and Visualization: After creating copies of the log files, Elasticsearch will index these logs and display their contents on Kibana graphical interface provided by Elastic Stack with data views following standard Mappings. Each mapping provides unique fields for each log data. After the visualization, we identify the message field that contains the logs provided by Tetragon to analyze the fields and values in this message.

As a result of the previous procedures, the data is indexed and ready for normalization. The normalization phase requires specific mappings and pipelines to extract important information from Tetragon's message. These pipelines are based on Elastic Common Schemas, using processors and conditions provided by Elasticsearch to reduce the complexity of the collected logs and make them in a readable format. The ingest pipeline consists of a series of configurable tasks called processors. Each processor runs sequentially, making specific changes to our incoming documents. After the processors have run, Elasticsearch adds the transformed documents to the data stream.

4.3. Threat Detection

This phase is the most essential phase, it involves the application of detection rules that are developed based on the scenarios discussed in section 5. These rules are enabled by the data normalization phase, which provides a consistent and reliable foundation for data analysis. The

Detection Engine component of Elasticsearch uses these rules to examine the indexed data and generate alerts for any anomalous or malicious behavior.

5. Results and experiments

Deploy Kubernetes Goat

Starting by deploying Kubernetes Goat which is a learning platform that offers more than 20 realistic scenarios. These scenarios include attacks, defenses, best practices, tools, and others. Table 5 shows some scenarios offered by Kubernetes Goat and how each scenario is categorized according to its security. Once the Kubernetes GOAT is deployed and attacking scenarios are explored, it proceeds to the next step, which is data collection.

Category	Scenarios
Attack	DIND (docker-in-docker) exploitation. Container escape to the host system. Attacking private registry. RBAC Least Priv misconfiguration. Kubernetes namespaces bypass.
Defense	Falco - Runtime security monitoring & detection. Cilium Tetragon - eBPF-based Security Observability and Runtime Enforcement. Securing Kubernetes Clusters using Kyverno Policy Engine.
Best Practices	Secure Network Limits Using NSP. Securing Kubernetes Clusters using Kyverno Policy Engine.

Table 2
Kubernetes Goat app scenarios and categories [34]

Data Collection

After deploying Tetragon, the initial step involves rolling it out and then activating the feature that allows the modifications of capability and namespace through the config map. This can be accomplished by updating the values of "enable process-cred" and "enable-process-ns" from "false" to "true." The File Access tracing policies can be activated by applying a YAML configuration file. This file specifies a tracing policy for the CNI cilium (Container Network Interface) plugin, which is used by Kubernetes. The tracing policy enables to monitor the system calls executed by the processes running on a node within the Kubernetes cluster. The policy specifies four kprobes that will be used to trace events related to file operations.

The network observability tracing policies can also be activated by applying a YAML file that defines a Cilium TracingPolicy for network connections. This policy indicates certain kprobes that will be employed to trace events related to TCP connections.

A YAML file defines a Cilium tracing policy for network connections. The policy specifies three kprobes that will be used to trace events related to TCP connections. The kprobes are:

- **tcp_connect** which is triggered when a TCP connection is established. The first argument of this kprobe is a sock structure that contains information about the connection.
- **tcp_close** which is triggered when a TCP connection is closed. The first argument of this kprobe is also a sock structure that contains information about the connection.
- **tcp_sendmsg** which is triggered when a TCP message is sent. The first argument of this kprobe is a sock structure that contains information about the connection. The second argument is an integer that represents the length of the message.

To locate the stored logs from Tetragon, access to the cluster files is required. This involves finding the Docker container that hosts this cluster. After finding the container that hosts the cluster files, the following steps involve installing the Elastic Agent and configuring the Fleet server. These steps enable the integration and management of the Elastic Agent within the cluster. The Elastic Agent is a lightweight data shipper that collects and forwards logs and metrics to the Fleet server for centralized management and analysis.

Custom Logs Integration Configuration

By adding Custom Logs to the Fleet Server Policy, The path from which Elastic-Agent should retrieve the data that can be specified easily. In this case, the path is used from the cluster files.

Check Index Data in Elastic

Running several containerized processes for multiple days, allowing Tetragon to capture their logs. Elastic agent should retrieve these logs continuously without any disruption. The logs are then indexed by Elasticsearch and displayed on Kibana's dashboards as a data view.

Analyse Message Field

The objective is to visually analyze the message field, which contains Tetragon-generated logs in JSON format. These logs contain a variety of fields and values related to container-to-host kernel processes and events. The main aim of this analysis is to enhance the data by adding new fields following the Elastic Common Schema (ECS) to standardize and normalize it. This involves extracting field names and their expected value formats from the messages.

Create and Configure Pipeline

Extracting pertinent details from Tetragon's message fields and content involves employing specialized processors to convert Tetragon logs into a human-readable format. This process is facilitated through a designated pipeline called "logs-tetra-default," where suitable processors are chosen for each field and data type or format to attain the intended outcomes. Additionally, normalizing log files with Elastic Common Schema (ECS) necessitates the incorporation of mandatory ECS fields like "Event. Type" and "Event.Category." While Tetragon logs may not inherently produce these fields, they can be derived from certain log values following a comprehensive analysis.

To apply the pipeline to the current data stream, the Custom logs integration for the Fleet server policy is configured Table 3 lists the required processors for the normalization process:

Processor	Value
Dissect	Splits a field into multiple fields using a delimiter.
JSON	Parses a JSON string and adds it as a new object field.
Rename	Renames an existing field to a new name.
Split	Splits a field into an array using a separator character.
Set	Sets the value of a field to a predefined value.
Script	Executes a script to modify documents before indexing.
Remove	Removes one or more fields from the document.

Table 3
Used Processors in the pipeline.

Check Data Quality

After applying the normalization pipeline, a powerful feature in Kibana called Data Quality is utilized to analyze the data view field types and compare them to the expected values and data types of the Elastic Common Schema. Any incorrect value indicates the need for pipeline maintenance or additional mapping if necessary. Results show the total number of fields available, the number of fields that are compliant with ECS (Elastic Common Schema), and The number of fields that have been customized. Using Data Quality in Kibana helps ensure data accuracy and consistency, identifying any discrepancies in the field types and assisting in maintaining a well-structured and standardized data representation.

Threat Detection

Before configuring detection rules, test Kubernetes Goat App's attack capabilities on scenarios like container escape and namespace bypass, as outlined in Section 5. One scenario demonstrates detecting privilege escalation attacks using Tetragon, where a container escape leads to host system access. The process involves accessing the system-monitor pod with a specific command, then exploiting it with "nsenter."

Another critical scenario involves attacking the private registry, explained in Section 5. This entails executing a set of commands to obtain the 'k8s-goat-FLAG' flag value in private registry images. These commands allow exploration and retrieval of container image information, including registry details, image catalog, manifests, and specific properties like environment variables, essential for Docker and container management tasks.

Create Detection Rules

By exploiting the attacks in the previous task, rules for the detection of these attacks were created using Kibana's ability to form various types of rules such as queries using KQL and EQL. The source of the data that the rule should apply to (logs-generic-*) was defined, along with the query in the chosen query language, and details about the rule such as severity and risk score. A schedule was then added to the rule to run at specific time intervals, along with an action to be taken when an attack is detected, such as sending alerts.

For each scenario selection in the previous task, a detection rule was created. The detection query for the "Container escape to the host" attack is written in EQL.

```
    process where
process.args == "execve rootcwd
clone" and
process.parent.executable
== "/usr/local/sbin/runc"
and
process.parent.args == "execve
clone" and
(process.executable == "/bin/sh"
 or
 process.executable == "/bin/bash")
```

The detection query for the container private registry attack is written in KQL.

```
event.category :"process" and
process.executable :"/bin/registry"
and process.args : "execve"
```

This query is looking for events where the category is "process", the executable involved is "/bin/registry", and one of the arguments in the process is "execve", which is a system call in Unix-like operating systems that replaces the current process image with a new one.

After each detection, an alert will be created and displayed in Kibana which provides several ways of alerting for an improved alerting system. The upcoming task will implement the configuration of these alerts.

Configure and test Alerts

Kibana's email connector is selected for our alerting system, allowing the generation of email alerts when specific conditions are met. Configuration includes sender information and rule settings, including recipient and message content. Testing involves triggering attacks to verify the alert function. Subsequently, swift action is taken to address and mitigate security incidents.

6. Discussion

Our proposal stands as a comprehensive solution aimed at solving the challenge of enhancing threat detection capabilities within containerized environments. By promoting a powerful interaction between visibility tools and log analysis processes, our approach empowers Security Operations Center (SOC) Analysts. It enables them to efficiently identify and respond to threats or malicious activities, ensuring a robust defense against intrusion in real time within the container runtime environment.

The findings of this study clearly show that the integration of Cilium Tetragon and Elastic Stack significantly enhances the threat detection capabilities within containerized systems. By employing readable forms for writing detection rules, this study contributes to the improvement of runtime security in Cloud native environment. Simplifying log analysis enables SOC analysts to readily identify abnormal or malicious activities.

One explanation for the success of this approach could be the combination of powerful visibility tools and simplified log analysis techniques, which augment the traditional capabilities of SOC analysts. This integration enables not only real-time monitoring but also proactive threat mitigation, addressing the limitations posed by traditional security measures in Cloud native environments. This study takes a strong position that an integrated approach, as suggested, is crucial for improving the runtime security of Cloud native environments.

Therefore, in the current landscape where security threats evolve at an unprecedented pace, implementing a solution that provides real-time visibility and actionable insights has become an imperative, not just an option. This study, while providing a foundational approach to runtime security, had its limitations in terms of its focus on specific tools—namely Cilium Tetragon and Elastic Stack. it explored how the integration of these specific tools can be leveraged to strengthen the security measures in Cloud native environments. Although these tools were effective for the purposes of this study, it's worth noting that the ever-evolving cybersecurity landscape necessitates continuous research and adaptation. Therefore, the study serves as a starting point for future research.

Future research could explore alternative tool combinations and metrics for comparing threat detection strategies. Long-term monitoring would provide insights into the solution's adaptability. As Cloud-native architectures evolve, our security approach must adapt too. This study enhances threat detection, equipping security professionals for timely action, and serves as a foundational improvement in runtime security.

7. Conclusion

The growing adoption of Cloud native architectures offers efficiency and scalability but presents security challenges. This paper presents a case study for robust security in Cloud-native environments, integrating Cilium Tetragon and Elastic Stack for real-time threat detection. It serves as a proof of concept for future container runtime security research. Traditional security approaches in Cloud-native settings are inadequate, so we aim to enhance threat detection. Future work includes expanding our resource index for network and endpoint visibility and deploying machine learning for unknown threat detection.

Acknowledgment

We would like to express our sincere gratitude to all individuals and the Octodet team who have contributed to the completion of this project. Their support, guidance, and assistance have been invaluable. This work was partially supported by the LABEX-TA project MeFoGL: "Méthode Formelles pour le Génie Logiciel"

References

[1] B. Scholl, T. Swanson, P. Jausovec, Cloud Native: Using Containers, Functions, and Data to Build Next-Generation Applications, " O'Reilly Media, Inc.", 2019.

[2] M. Ammi, O. Adedugbe, F. M. Alharby, E. Benkhelifa, Leveraging a cloud-native architecture to enable semantic interconnectedness of data for cyber threat intelligence, Cluster Computing 25 (2022) 3629–3640.

[3] E. Casalicchio, Container orchestration: A survey, Systems Modeling: Methodologies and Tools (2019) 221–235.

[4] The 4c's of cloud native security, 2023. [Online]. Available from: https://www.linkedin.com/pulse/4cs-Cloud-native-security-tl-consulting-group/.

[5] M. Luksa, Kubernetes in action, Simon and Schuster, 2017.

[6] R. Vettor, S. Smith, Architecting cloud-native .net apps for azure, Washington: Microsoft Corporation (2022).

[7] E. Preeth, F. J. P. Mulerickal, B. Paul, Y. Sastri, Evaluation of docker containers based on hardware utilization, in: 2015 international conference on control communication & computing India (ICCC), IEEE, 2015, pp. 697–700.

[8] G. Bhatia, A. Choudhary, V. Gupta, The road to docker: a survey, International Journal of Advanced Research in Computer Science 8 (2017) 83–87.

[9] Developer survey, 2023. [Online]. Available from: urlhttps://survey.stackoverflow.co/2023/.

[10] S. Sultan, I. Ahmad, T. Dimitriou, Container security: Issues, challenges, and the road ahead, IEEE Access 7 (2019) 52976–52996. doi:10.1109/ACCESS.2019.2911732.

[11] E. Casalicchio, Container orchestration: A survey, Systems Modeling: Methodologies and Tools (2019) 221–235.

[12] I. C. Education, What are ci/cd and the ci/cd pipeline?, 2021. [Online]. Available from: https://www.ibm.com/blog/ci-cd-pipeline/.

[13] Serverless cncf glossary, 2022. [Online]. Available from: https://glossary.cncf.io/serverless/.

[14] C. Native, Overview of cloud native security, 2023. [Online]. Available from: https://kubernetes.io/docs/concepts/security/overview/.

[15] B. Sosinsky, Cloud computing bible, volume 762, John Wiley & Sons, 2010.

[16] O. Flauzac, F. Mauhourat, F. Nolot, A review of native container security for running applications, Procedia Computer Science 175 (2020) 157–164.

[17] M. Reeves, D. J. Tian, A. Bianchi, Z. B. Celik, Towards improving container security by preventing runtime escapes (2021) 38–46.

[18] G. Pai, 7 key features for kubernetes and container security, 2023. [Online]. Available from: https://www.infoworld.com/article/3699109/7-key-features-for-kubernetes-and-container-security.html.

[19] G. Fournier, S. Afchain, S. Baubeau, Runtime security monitoring with ebpf, in: 17th SSTIC Symposium sur la Sécurité des Technologies de l'Information et de la Communication, 2021.

[20] C. Cassagnes, L. Trestioreanu, C. Joly, R. State, The rise of ebpf for non-intrusive performance monitoring, in: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2020, pp. 1–7.

[21] D. Calavera, L. Fontana, Linux Observability with BPF: Advanced Programming for Per-

formance Analysis and Networking, O'Reilly Media, 2019.

[22] M. N. Al-Mhiqani, R. Ahmad, Z. Zainal Abidin, W. Yassin, A. Hassan, K. H. Abdulkareem, N. S. Ali, Z. Yunos, A review of insider threat detection: Classification, machine learning techniques, datasets, open challenges, and recommendations, Applied Sciences 10 (2020) 5208.

[23] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, Q. Zhou, A measurement study on linux container security: Attacks and countermeasures, in: Proceedings of the 34th Annual Computer Security Applications Conference, 2018, pp. 418–429.

[24] S. Yang, D. Yan, G. Xu, A. Rountev, Dynamic analysis of inefficiently-used containers, in: Proceedings of the Ninth International Workshop on Dynamic Analysis, 2012, pp. 30–35.

[25] S. R. K. Patil, N. John, P. S. Kunja, A. Dwivedi, S. Suganthi, P. B. Honnnavali, Hardening containers with static and dynamic analysis, in: Proceedings of the International Conference on Cybersecurity, Situational Awareness and Social Media: Cyber Science 2022; 20–21 June; Wales, Springer, 2023, pp. 207–227.

[26] F. Sabahi, A. Movaghar, Intrusion detection: A survey, in: 2008 Third International Conference on Systems and Networks Communications, 2008, pp. 23–26. doi:10.1109/ICSNC.2008.44.

[27] S. Neupane, J. Ables, W. Anderson, S. Mittal, S. Rahimi, I. Banicescu, M. Seale, Explainable intrusion detection systems (x-ids): A survey of current methods, challenges, and opportunities, IEEE Access 10 (2022) 112392–112415.

[28] A. Thakkar, R. Lohiya, A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions, Artificial Intelligence Review 55 (2022) 453–563.

[29] O. Tunde-Onadele, J. He, T. Dai, X. Gu, A study on container vulnerability exploit detection, in: 2019 ieee international conference on Cloud engineering (IC2E), IEEE, 2019, pp. 121–127.

[30] V. Bandari, A comprehensive review of ai applications in automated container orchestration, predictive maintenance, security and compliance, resource optimization, and continuous deployment and testing, International Journal of Intelligent Automation and Computing 4 (2021) 1–19.

[31] A. J. M. Editor, Advanced sciences and technologies for security applications security by design innovative perspectives on complex problems, ???? URL: http://www.springer.com/series/5540.

[32] Cilium tetragon, 2023. [Online]. Available from: https://isovalent.com/tetragon/.

[33] Container workload protection | elastic security solution, 2023. [Online]. Available from: https://www.elastic.co/guide/en/security/master/d4c-overview.html.

[34] Kubernetes goat scenarios, 2023. [Online]. Available from: https://madhuakula.com/kubernetes-goat/docs/scenarios.