iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

SECURING AUTONOMOUS AI AGENTS: DISTRIBUTED, SANDBOXED EXECUTION ENVIRONMENTS VIA WEBASSEMBLY AND KUBERNETES

Prem Pradeep Motgi

ABSTRACT
The emergence of Agentic Artificial Intelligence (AI) marks a significant shift from passive generative systems to autonomous agents capable of reasoning, planning, and executing actions with minimal human intervention. These agents increasingly interact with external tools, APIs, cloud resources, and software environments, enabling advanced automation across domains such as software engineering, cybersecurity, business operations, and scientific research. However, the ability of AI agents to generate and execute code autonomously introduces substantial security challenges, including unauthorized resource access, privilege escalation, prompt injection attacks, malicious code execution, data leakage, and supply chain vulnerabilities. Traditional security mechanisms designed for human-operated applications are often insufficient to address the dynamic and autonomous nature of agent-driven execution environments.
This study proposes a distributed and sandboxed execution architecture for securing autonomous AI agents through the integration of WebAssembly (Wasm), container runtimes, and Kubernetes-based orchestration. The proposed framework adopts a defense-in-depth approach that isolates AI-generated actions within lightweight Wasm sandboxes while leveraging Kubernetes for scalable workload management, policy enforcement, resource governance, and runtime monitoring. By combining cloud-native technologies with secure execution principles, the architecture aims to minimize attack surfaces, contain potentially harmful agent behaviors, and provide auditable execution pathways for autonomous operations.
A design science research methodology is employed to develop and evaluate the conceptual framework. The architecture is analyzed against common threat scenarios associated with Agentic AI, including code injection, unauthorized system interactions, and compromised execution modules. The findings indicate that WebAssembly-based sandboxing offers stronger isolation and reduced overhead compared to traditional virtualized environments, while Kubernetes enhances scalability and operational resilience. The study contributes a vendor-neutral security model for autonomous AI systems and provides practical guidance for organizations seeking to deploy trustworthy, secure, and scalable Agentic AI infrastructures. Future research directions include confidential computing integration, adaptive policy engines, and decentralized security frameworks for multi-agent ecosystems.

Keywords:
Agentic AI, Autonomous AI Agents, WebAssembly, Kubernetes, Sandboxed Execution, Cloud-Native Security, AI Governance, Secure AI Systems.

1. INTRODUCTION

The rapid advancement of artificial intelligence (AI) has transformed the capabilities of intelligent systems from passive content generation to autonomous decision-making and task execution. Recent developments in Agentic AI have enabled AI systems to independently plan, reason, interact with external tools, and execute actions with limited human supervision. Unlike conventional large language models that primarily generate text responses, autonomous AI agents can invoke application programming interfaces (APIs), manipulate files, execute code, manage workflows, and coordinate complex tasks across distributed environments. This evolution has created significant opportunities for automation in software engineering, cybersecurity operations, business process management, scientific research, and cloud computing. However, the increasing autonomy of AI agents also introduces unprecedented security challenges that require robust execution controls and isolation mechanisms.
A fundamental concern in Agentic AI systems is the ability of agents to generate and execute code autonomously. While autonomous code execution enhances productivity and operational efficiency, it simultaneously expands the attack surface of computing environments. Malicious or unintended actions generated by AI agents may result in unauthorized system access, privilege escalation, data exfiltration, denial-of-service attacks, or the execution of vulnerable software components. These risks become more severe in cloud-native environments where autonomous agents interact with critical infrastructure, enterprise systems, and external services. Consequently,

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

ensuring secure execution environments has emerged as one of the most significant barriers to the widespread deployment of autonomous AI agents.

Traditional security approaches such as virtual machines and container-based isolation have been widely adopted to mitigate risks associated with untrusted code execution. Containers offer lightweight virtualization and efficient resource utilization, making them attractive for large-scale deployment environments. However, containerized applications share the host operating system kernel, creating potential attack vectors that can be exploited through kernel vulnerabilities or excessive system call privileges. Ghafari et al. (2019) demonstrated that restricting container access through fine-grained sandboxing significantly reduces attack surfaces and improves protection against container-based attacks. Similarly, Kim et al. (2024) proposed dynamic system call filtering mechanisms that further strengthen container security by restricting unnecessary kernel interactions during runtime. These studies highlight the importance of runtime isolation and access control in securing modern execution environments.

Despite advances in container security, emerging Agentic AI systems demand stronger guarantees for executing dynamically generated and potentially untrusted code. In response to these challenges, WebAssembly (Wasm) has gained significant attention as a secure and lightweight execution technology capable of providing fine-grained sandboxing and platform-independent deployment. Originally developed for web applications, WebAssembly has evolved into a general-purpose runtime environment suitable for cloud-native workloads and secure code execution. According to Bazzani et al. (2025), WebAssembly provides strong security properties through memory isolation, controlled execution environments, and restricted access to host resources. These characteristics make it a promising candidate for securing autonomous AI agents that require dynamic code execution capabilities.

Recent research has further strengthened the security foundations of WebAssembly. Watt et al. (2022) demonstrated that WebAssembly can support provably safe multilingual software sandboxing, enabling secure execution of code compiled from multiple programming languages. Likewise, Sjösten et al. (2023) introduced SecWasm, a framework that enhances WebAssembly security through information flow control mechanisms, thereby reducing the risk of unauthorized data leakage between execution contexts. Additional security enhancements have been proposed through stack protection mechanisms such as WASP, which strengthens runtime defenses against memory-related attacks within WebAssembly environments (Di Federico et al., 2026). Furthermore, vulnerability analysis tools such as Wasmati have been developed to identify security weaknesses in WebAssembly modules before deployment, contributing to more secure execution pipelines (Lehmann et al., 2022). Hallak et al. (2024) also highlighted the growing body of research dedicated to WebAssembly analysis techniques, emphasizing its increasing role in secure cloud-native computing.

While secure code execution is a critical component of Agentic AI security, large-scale deployment additionally requires efficient orchestration, resource management, and policy enforcement mechanisms. Kubernetes has emerged as the dominant platform for orchestrating containerized and cloud-native applications across distributed environments. Its scalability, resilience, and automation capabilities make it a suitable foundation for managing autonomous AI workloads. However, Kubernetes environments introduce their own security challenges. Research by Zhang et al. (2025) revealed significant vulnerabilities associated with Kubernetes control plane interfaces, demonstrating how insufficient access controls can expose critical infrastructure to attacks. Similarly, Aljuhani et al. (2025) emphasized the need for enhanced monitoring and behavioral analysis mechanisms to detect and mitigate threats targeting Kubernetes clusters. These findings indicate that orchestration platforms must be carefully secured when serving as execution environments for autonomous AI agents.

The convergence of Agentic AI, WebAssembly, and Kubernetes presents a unique opportunity to develop a comprehensive security architecture capable of addressing the risks associated with autonomous code execution. By combining WebAssembly-based sandboxing with Kubernetes-driven orchestration, organizations can create distributed execution environments that isolate AI-generated actions while maintaining scalability, performance, and operational flexibility. Such an approach aligns with modern cloud-native security principles, including zero-trust architecture, least-privilege access control, runtime monitoring, and defense-in-depth strategies.

Despite significant progress in WebAssembly security and Kubernetes orchestration, existing research remains fragmented, with limited studies exploring their combined application to Agentic AI systems. Most prior work focuses either on secure container environments, WebAssembly runtime security, or cloud-native orchestration independently. Consequently, there remains a critical research gap regarding the design of integrated architectures capable of securely executing autonomous AI-generated actions within distributed environments. Addressing this gap is essential as organizations increasingly adopt AI agents for mission-critical operations that demand both autonomy and security.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Therefore, this study proposes a distributed and sandboxed execution architecture for securing autonomous AI agents through the integration of WebAssembly runtimes, container technologies, and Kubernetes orchestration. The proposed framework seeks to provide secure isolation, controlled resource access, scalable deployment, and comprehensive monitoring for AI-generated actions. By leveraging established cloud-native technologies and modern security principles, the study aims to contribute a vendor-neutral approach for building trustworthy Agentic AI infrastructures capable of supporting future autonomous computing ecosystems.

Specifically, the objectives of this study are to examine the security challenges associated with autonomous AI agent execution, evaluate the role of WebAssembly as a secure sandboxing technology, analyze the security implications of Kubernetes-based orchestration, and propose a unified architecture that integrates these technologies to mitigate execution-related risks. The findings are expected to provide valuable insights for researchers, cloud architects, security practitioners, and organizations seeking to deploy secure, scalable, and resilient Agentic AI systems in increasingly complex digital environments.

2. LITERATURE REVIEW

2.1 Overview of Autonomous Agent Systems

The development of autonomous software agents represents a significant evolution in modern computing systems. Unlike conventional applications that operate according to predefined instructions, autonomous agents possess the capability to perceive environmental inputs, reason about available information, formulate plans, and execute actions independently. These systems increasingly interact with external tools, cloud services, application programming interfaces (APIs), databases, and software environments to accomplish complex objectives. Such capabilities have expanded the potential applications of autonomous agents across software development, cybersecurity operations, business process automation, scientific computing, and cloud infrastructure management.

As the operational autonomy of software agents increases, so does the complexity of securing their execution environments. Autonomous systems frequently generate executable commands, scripts, and workflows that interact directly with sensitive computational resources. Consequently, ensuring that these actions occur within controlled and isolated environments has become a critical requirement for maintaining system integrity, confidentiality, and availability.

2.2 Security Challenges in Autonomous Code Execution

The ability of autonomous agents to execute dynamically generated actions introduces security concerns that differ substantially from those associated with traditional software applications. Since generated instructions may not undergo the same validation procedures as manually developed software, execution environments must be capable of containing unexpected or potentially harmful behavior.

Several attack vectors have been identified within modern autonomous computing environments. These include unauthorized resource access, privilege escalation, execution of malicious code, information leakage, supply-chain compromises, and exploitation of runtime vulnerabilities. The increasing reliance on cloud-native infrastructures further amplifies these risks because compromised workloads can potentially affect interconnected services operating within the same ecosystem.

Traditional security mechanisms such as access control policies and network segmentation provide partial protection; however, they are often insufficient when dealing with dynamically generated workloads. Consequently, research has increasingly focused on secure sandboxing, runtime isolation, and policy-based execution control as mechanisms for reducing the attack surface associated with autonomous operations.

2.3 Sandboxing and Container-Based Isolation

Sandboxing has emerged as one of the most widely adopted approaches for restricting the capabilities of untrusted code. By isolating applications from critical system resources, sandboxing mechanisms limit the potential impact of compromised or malicious workloads.

Container technology has become particularly influential within modern cloud computing environments due to its lightweight architecture and efficient resource utilization. Containers enable applications to operate within isolated execution contexts while sharing the host operating system kernel. Although this model improves scalability and deployment efficiency, it also introduces security concerns associated with shared-kernel architectures.

Ghafari et al. (2019) investigated practical approaches to container sandboxing and demonstrated that restricting container access to system calls can significantly reduce attack surfaces. Their findings revealed that fine-grained sandbox enforcement effectively limits opportunities for attackers to exploit kernel interfaces while maintaining

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

acceptable performance overhead. The study highlighted the importance of behavior-based restriction mechanisms in strengthening container security.

Building upon this concept, Kim et al. (2024) proposed Optimus, a dynamic system call filtering framework designed to reduce unnecessary interactions between containers and the host operating system. The framework utilizes association-based analysis to identify and restrict system calls that are not essential for container functionality. Experimental results demonstrated that dynamic filtering can enhance security without significantly affecting service availability. These findings reinforce the view that runtime restriction mechanisms play a crucial role in protecting cloud-native execution environments.

Despite these advancements, containers remain susceptible to vulnerabilities arising from kernel-level exploits, privilege misconfigurations, and shared-resource dependencies. As a result, researchers have increasingly explored alternative execution technologies capable of providing stronger isolation guarantees while preserving operational efficiency.

2.4 WebAssembly as a Secure Execution Environment

WebAssembly (Wasm) has gained considerable attention as a lightweight, portable, and secure execution technology suitable for cloud-native environments. Initially developed to enable high-performance web applications, WebAssembly has evolved into a general-purpose runtime platform capable of supporting secure execution across diverse computing environments.

According to Bazzani et al. (2025), WebAssembly offers several security advantages, including memory isolation, restricted host interaction, deterministic execution behavior, and reduced attack surfaces. These characteristics make WebAssembly particularly attractive for environments that require secure execution of untrusted code. Unlike traditional containers, WebAssembly modules operate within tightly controlled sandboxes that restrict direct access to system resources unless explicitly authorized.

The security architecture of WebAssembly has motivated extensive research into formal verification and secure execution guarantees. Watt et al. (2022) demonstrated that WebAssembly can support provably safe multilingual software sandboxing. Their work showed that carefully designed WebAssembly runtimes can enforce strong isolation properties while maintaining competitive performance characteristics. This capability is particularly valuable in heterogeneous computing environments where code originating from multiple programming languages must be executed securely.

Further advancements have focused on improving information security within WebAssembly environments. Sjösten et al. (2023) introduced SecWasm, an information flow control framework that strengthens confidentiality guarantees by preventing unauthorized data transfers between execution contexts. By incorporating formal information flow policies, SecWasm addresses one of the fundamental challenges associated with secure execution environments: preventing unintended information leakage.

Additional security enhancements have been proposed at the runtime level. Di Federico et al. (2026) developed WASP, a stack protection mechanism designed to strengthen WebAssembly defenses against memory corruption attacks. Their research demonstrated that enhanced stack protection can improve runtime resilience while preserving execution efficiency. Such developments indicate a growing emphasis on strengthening the security foundations of WebAssembly beyond its native sandboxing capabilities.

Despite its security benefits, WebAssembly is not immune to vulnerabilities. Lehmann et al. (2022) developed Wasmati, a static vulnerability scanning framework that identifies security weaknesses within WebAssembly modules prior to deployment. Their findings revealed that vulnerabilities may still arise from insecure coding practices, logic errors, and implementation flaws. Consequently, secure execution requires not only runtime isolation but also rigorous code analysis and validation processes.

Hallak et al. (2024) conducted a systematic review of WebAssembly analysis techniques and highlighted the rapid expansion of research dedicated to security assessment, vulnerability detection, and formal verification. Their review demonstrated that WebAssembly has matured into a significant research area within secure computing and cloud-native infrastructure, although challenges related to tooling, standardization, and ecosystem maturity remain active areas of investigation.

2.5 Kubernetes and Distributed Orchestration Security

As modern applications increasingly rely on distributed architectures, orchestration platforms have become essential for managing workloads across large-scale computing environments. Kubernetes has emerged as the dominant orchestration framework due to its scalability, resilience, and automation capabilities.

Kubernetes provides mechanisms for workload scheduling, resource allocation, service discovery, and automated recovery. These features make it particularly suitable for managing distributed execution environments where

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

workloads must be deployed dynamically and efficiently. However, the growing adoption of Kubernetes has also attracted significant attention from security researchers seeking to understand and mitigate emerging threats.

Zhang et al. (2025) examined security weaknesses associated with Kubernetes control plane interfaces and identified several vulnerabilities related to access control deficiencies and exposed management functions. Their findings demonstrated that misconfigured control plane components can enable attackers to gain unauthorized access, disrupt services, and compromise sensitive resources. The study emphasized the necessity of implementing robust security controls within orchestration environments.

Complementing this perspective, Aljuhani et al. (2025) proposed Shadowkube, a security framework that integrates behavioral monitoring and honeypot technologies within Kubernetes environments. The framework improves threat detection capabilities by identifying suspicious activities and collecting intelligence on potential attackers. Experimental evaluations demonstrated that proactive monitoring mechanisms can enhance the security posture of distributed cloud-native systems.

These studies collectively suggest that while Kubernetes provides powerful orchestration capabilities, securing the orchestration layer remains essential for protecting distributed execution environments. Effective deployment therefore requires a combination of access control, runtime monitoring, behavioral analysis, and policy enforcement mechanisms.

2.6 Research Gap

The reviewed literature demonstrates significant progress in the areas of container security, WebAssembly-based sandboxing, runtime protection mechanisms, vulnerability analysis, and Kubernetes security. Existing studies have established that container sandboxing can reduce attack surfaces, WebAssembly can provide stronger isolation guarantees, and Kubernetes can support scalable orchestration of distributed workloads. However, most research efforts examine these technologies independently rather than as components of a unified security architecture.

Current literature provides limited guidance regarding the integration of WebAssembly runtimes, container technologies, and Kubernetes orchestration for securing autonomous execution environments. Furthermore, relatively few studies investigate how these technologies can collectively address the unique challenges associated with dynamically generated code execution, runtime isolation, policy enforcement, and large-scale workload management.

This gap is particularly important as autonomous software systems increasingly require secure environments capable of executing untrusted actions while maintaining scalability and operational flexibility. Consequently, there is a need for a comprehensive framework that combines WebAssembly-based sandboxing with Kubernetes-driven orchestration to provide secure, distributed, and vendor-neutral execution environments. Addressing this gap forms the foundation of the proposed architecture presented in this study.

Table 2.1 Overview of Autonomous Agent Systems and Their Characteristics

Agent Characteristic	Description	Functional Role	Security Implications
Autonomy	Ability to perform tasks independently without continuous human intervention	Enables automated decision-making and task execution	May execute unintended or unauthorized actions if controls are insufficient
Perception	Capability to gather information from internal and external environments	Supports situational awareness and context understanding	Exposure to manipulated or malicious inputs can affect decision quality
Reasoning	Ability to analyze information and determine appropriate actions	Facilitates planning, problem-solving, and decision-making	Incorrect reasoning may lead to unsafe or undesirable outcomes
Planning	Generation of action sequences to achieve specific objectives	Improves efficiency in complex task execution	Poorly constrained plans may access sensitive resources
Adaptability	Capacity to modify behavior based on changing conditions	Enhances resilience and operational flexibility	Dynamic behavior can create unpredictable security risks

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Tool Integration	Interaction with APIs, databases, software applications, and cloud services	Extends agent functionality beyond internal capabilities	Increases attack surface through external dependencies
Communication	Exchange of information with users, systems, or other agents	Supports collaboration and distributed operations	Unauthorized communication channels may facilitate data leakage
Learning Capability	Improvement of performance through feedback and experience	Enables continuous optimization and adaptation	Learning from compromised data can introduce vulnerabilities
Distributed Operation	Execution across multiple systems, nodes, or cloud environments	Supports scalability and fault tolerance	Expands security boundaries and complexity of management
Action Execution	Direct implementation of generated commands or workflows	Allows completion of real-world tasks	Requires strict isolation and monitoring mechanisms

Table Caption: Table 2.1 presents the core characteristics of autonomous agent systems, their functional roles in distributed computing environments, and the associated security implications that necessitate secure execution architectures. This overview provides the conceptual foundation for understanding the security requirements addressed by WebAssembly-based sandboxing and Kubernetes orchestration in subsequent sections.

3. METHODOLOGY

3.1 Research Design

This study adopts a Design Science Research (DSR) methodology to develop and evaluate a secure execution architecture for autonomous agent systems operating in distributed cloud-native environments. Design Science Research is widely used in information systems and computer science to address practical problems through the creation of innovative artifacts, frameworks, models, and architectures. The methodology is particularly suitable for this study because the primary objective is not to test an existing theory but to design and propose a security framework capable of mitigating risks associated with autonomous code execution.

The research follows a conceptual and architecture-driven approach that integrates existing knowledge from secure software execution, WebAssembly technologies, container security, and cloud orchestration platforms. Through a systematic analysis of current security challenges and existing solutions, the study develops a vendor-neutral framework that combines WebAssembly sandboxing, container runtime technologies, and Kubernetes orchestration to establish secure execution boundaries for autonomous computational agents.

The proposed framework is evaluated through theoretical security analysis, comparative assessment with existing execution environments, and threat-based validation. This approach enables the examination of how different architectural components contribute to reducing attack surfaces, enforcing isolation policies, and improving overall execution security.

3.2 Proposed Security Framework

The proposed architecture consists of six interconnected layers designed to provide secure, scalable, and auditable execution environments for autonomous workloads. Each layer contributes specific security functions that collectively establish a defense-in-depth architecture.

3.2.1 Agent Layer

The Agent Layer represents the entry point of the framework and contains autonomous agents responsible for task planning, decision-making, workflow generation, and action execution. Agents receive requests from users or external systems and generate execution instructions required to accomplish assigned objectives.

Given the dynamic nature of generated actions, the Agent Layer does not directly interact with critical infrastructure resources. Instead, all generated instructions must pass through predefined validation and security controls before execution is permitted.

3.2.2 Policy Enforcement Layer

The Policy Enforcement Layer acts as a security gateway between autonomous agents and execution environments. This layer validates generated actions against predefined organizational security policies, access control rules, and operational constraints.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

The primary responsibilities of this layer include:

- ✓ Action validation.
- ✓ Resource authorization.
- ✓ Privilege restriction.
- ✓ Compliance verification.
- ✓ Execution approval.

By implementing policy-driven controls, the framework reduces the likelihood of unauthorized operations and limits opportunities for privilege escalation.

3.2.3 WebAssembly Sandbox Layer

The WebAssembly Sandbox Layer provides the primary execution environment for generated workloads. All executable actions are encapsulated within isolated WebAssembly modules before execution.

This layer delivers several security benefits:

- ✓ Memory isolation.
- ✓ Controlled host interaction.
- ✓ Reduced attack surface.
- ✓ Platform independence.
- ✓ Lightweight execution.

Because WebAssembly modules operate within restricted runtime environments, unauthorized access to system resources is prevented unless explicitly permitted through controlled interfaces.

3.2.4 Container Runtime Layer

The Container Runtime Layer provides an additional security boundary surrounding WebAssembly workloads. Technologies such as containerd are responsible for workload packaging, runtime management, and resource allocation.

This layer contributes:

- ❖ Process isolation.
- ❖ Resource governance.
- ❖ Runtime monitoring.
- ❖ Execution consistency.

The combination of container isolation and WebAssembly sandboxing establishes multiple layers of protection against execution-based attacks.

3.2.5 Kubernetes Orchestration Layer

The Kubernetes Orchestration Layer manages workload deployment, scheduling, scaling, and recovery across distributed computing environments. Kubernetes coordinates the execution of containerized workloads while enforcing operational policies and resource constraints.

Key functions include:

- ✓ Automated workload scheduling.
- ✓ Horizontal scalability.
- ✓ Service orchestration.
- ✓ Resource optimization.
- ✓ Fault tolerance.

Through centralized orchestration, the framework maintains consistent security policies across all execution nodes.

3.2.6 Monitoring and Audit Layer

The Monitoring and Audit Layer continuously collects operational and security-related information from all architectural components. This layer provides visibility into workload behavior and supports incident detection and forensic analysis.

Core capabilities include:

- ✓ Runtime monitoring.
- ✓ Event logging.
- ✓ Behavioral analysis.
- ✓ Threat detection.
- ✓ Compliance reporting.

Comprehensive monitoring ensures that suspicious activities can be identified and investigated in a timely manner.

3.3 System Architecture Design

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

The architecture follows a layered security model that separates decision-making components from execution environments. This separation minimizes direct exposure of critical resources and reduces the impact of compromised workloads.

3.3.1 Secure Request Lifecycle

The execution process begins when an external request is submitted to an autonomous agent. The agent analyzes the request and generates the necessary actions required to achieve the specified objective. Before execution, generated actions undergo policy validation and security assessment. Approved actions are then encapsulated within WebAssembly modules and executed inside isolated runtime environments managed by Kubernetes.

3.3.2 Agent Task Execution Workflow

The proposed workflow consists of the following stages:

- ✓ Task request submission.
- ✓ Agent reasoning and planning.
- ✓ Action generation.
- ✓ Policy validation.
- ✓ WebAssembly packaging.
- ✓ Sandbox execution.
- ✓ Runtime monitoring.
- ✓ Result verification.
- ✓ Audit logging.

This workflow ensures that all actions are subject to security controls before execution occurs.

3.3.3 Security Control Flow

The security control flow incorporates multiple checkpoints throughout the execution lifecycle. Each generated action is evaluated against security policies, monitored during execution, and recorded for audit purposes. This layered approach limits the ability of malicious or compromised workloads to bypass established controls.

3.4 Threat Modeling

Threat modeling is used to identify potential security risks affecting the proposed architecture and to evaluate corresponding mitigation strategies.

3.4.1 Threat Identification

The primary threats considered in this study include:

- ✓ Unauthorized code execution.
- ✓ Privilege escalation.
- ✓ Resource abuse.
- ✓ Information leakage.
- ✓ Runtime exploitation.
- ✓ Supply-chain attacks.
- ✓ Orchestration layer compromise.

These threats represent common attack vectors associated with distributed execution environments.

3.4.2 Attack Surface Analysis

The attack surface is analyzed across each architectural layer to identify possible entry points for adversaries. Particular attention is given to execution interfaces, runtime environments, orchestration mechanisms, and communication channels between system components.

The analysis evaluates how WebAssembly isolation, container boundaries, and Kubernetes security controls contribute to reducing exposure to identified threats.

3.4.3 Risk Classification

Threats are categorized according to their likelihood and potential impact on system confidentiality, integrity, and availability. High-risk threats receive priority consideration during architectural design and mitigation planning.

3.5 Evaluation Metrics

The effectiveness of the proposed framework is assessed using several security and operational metrics.

3.5.1 Isolation Effectiveness

Isolation effectiveness measures the ability of the architecture to prevent unauthorized interactions between workloads and protected resources. This metric evaluates the strength of WebAssembly sandboxing and container boundaries.

3.5.2 Runtime Overhead

Runtime overhead assesses the computational costs associated with implementing multiple security layers. The metric considers execution latency, memory consumption, and resource utilization.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

3.5.3 Scalability
Scalability evaluates the framework's ability to support increasing numbers of autonomous workloads without significant degradation in performance or security.

3.5.4 Security Resilience
Security resilience measures the framework's capability to withstand attacks, contain compromised workloads, and maintain operational continuity under adverse conditions.

3.5.5 Auditability and Observability
Auditability examines the extent to which execution activities can be monitored, traced, and investigated. Comprehensive visibility is essential for compliance, forensic analysis, and incident response.

3.6 Methodological Summary
The methodology combines Design Science Research principles with architecture-driven security analysis to develop a secure execution framework for autonomous computational systems. By integrating WebAssembly sandboxing, container runtime technologies, Kubernetes orchestration, and continuous monitoring, the proposed architecture establishes multiple layers of protection against execution-related threats. The framework provides a structured foundation for evaluating secure execution strategies within distributed cloud native environments and serves as the basis for the results and discussion presented in subsequent sections.

4. RESULTS

4.1 Proposed Architecture Overview
The proposed architecture was designed to address the security challenges associated with autonomous code execution in distributed cloud-native environments. The framework integrates WebAssembly-based sandboxing, container runtime isolation, Kubernetes orchestration, policy enforcement mechanisms, and continuous monitoring capabilities into a unified security model. The architectural analysis indicates that combining these technologies creates multiple layers of defense capable of reducing execution risks while maintaining scalability and operational flexibility.

The architecture separates autonomous decision-making components from execution environments through a layered security approach. Generated actions are subjected to policy validation before being encapsulated within WebAssembly modules and deployed through containerized execution environments orchestrated by Kubernetes. This separation minimizes direct access to critical infrastructure resources and reduces the potential impact of compromised workloads.

The evaluation demonstrates that each architectural layer contributes distinct security functions. The Policy Enforcement Layer provides access control and execution validation, the WebAssembly Sandbox Layer establishes workload isolation, the Container Runtime Layer manages resource governance, and the Kubernetes Layer ensures distributed workload orchestration and operational resilience.

4.2 Security Analysis Results

4.2.1 Execution Isolation Assessment
The security assessment indicates that the proposed framework provides enhanced workload isolation compared with conventional container-only execution environments. WebAssembly runtimes establish memory-safe execution boundaries that restrict direct interaction with host operating system resources unless explicitly authorized through controlled interfaces.

The dual-isolation model created by combining WebAssembly sandboxes with container runtimes significantly reduces opportunities for unauthorized resource access. Unlike traditional execution models where applications may have broader access to system resources, the proposed framework limits execution privileges to predefined policies and permitted runtime capabilities.

The analysis further reveals that layered isolation mechanisms improve containment of potentially harmful workloads. Even if a WebAssembly module exhibits unintended behavior, its actions remain constrained by both the runtime sandbox and the surrounding container environment.

4.2.2 Resource Access Restriction Analysis
The evaluation of resource access controls demonstrates that the framework effectively enforces the principle of least privilege. All generated actions are validated before execution and are granted only the permissions required to perform their designated functions.

Policy-based authorization reduces exposure to privilege escalation attacks by ensuring that workloads cannot access unauthorized files, services, or infrastructure components. The implementation of restricted execution environments further minimizes the attack surface available to malicious actors.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Results indicate that the combination of policy enforcement, WebAssembly sandboxing, and Kubernetes access controls establishes multiple checkpoints capable of preventing unauthorized interactions with protected resources.

4.2.3 Attack Containment Evaluation

The threat analysis suggests that the proposed framework effectively contains several categories of execution-related attacks. Potentially harmful workloads remain confined within isolated execution boundaries, limiting their ability to affect neighboring workloads or underlying infrastructure components.

The layered architecture demonstrates particular effectiveness against:

- ✓ Unauthorized code execution attempts.
- ✓ Resource abuse attacks.
- ✓ Runtime exploitation attempts.
- ✓ Information leakage scenarios.
- ✓ Lateral movement within distributed environments.

The containment capabilities of the framework significantly reduce the potential impact of security incidents while improving overall system resilience.

4.3 Performance Evaluation

4.3.1 Sandbox Deployment Performance

The performance assessment indicates that WebAssembly-based execution environments introduce minimal deployment overhead compared with traditional virtual machine architectures. Lightweight module deployment enables rapid workload initialization and efficient resource utilization.

The architecture benefits from the reduced startup times associated with WebAssembly runtimes, making the framework suitable for dynamic execution environments where workloads must be created and terminated frequently.

The findings suggest that the proposed framework can maintain strong security guarantees without imposing substantial performance penalties on operational workloads.

4.3.2 Kubernetes Scheduling Efficiency

The orchestration analysis demonstrates that Kubernetes effectively manages workload distribution across available computing resources. Automated scheduling mechanisms ensure balanced resource utilization while maintaining policy compliance and execution consistency.

Kubernetes contributes additional operational benefits, including:

- ❖ Automated workload recovery.
- ❖ Dynamic resource allocation.
- ❖ Horizontal scaling capabilities.
- ❖ High availability support.
- ❖ Fault-tolerant execution management.

These capabilities enhance the overall reliability of the proposed framework while supporting large-scale deployment scenarios.

4.3.3 Resource Utilization Analysis

The layered execution architecture achieves efficient resource utilization through lightweight WebAssembly runtimes and containerized deployment models. Compared with traditional virtualized infrastructures, the framework requires fewer computational resources while maintaining equivalent isolation objectives.

The evaluation indicates that resource consumption remains manageable even as workload volume increases, demonstrating the scalability potential of the architecture for enterprise deployment environments.

4.4 Comparative Analysis

4.4.1 Traditional Containers versus WebAssembly Sandboxes

The comparative assessment reveals notable differences between conventional container execution and WebAssembly-based sandboxing.

WebAssembly environments provide stronger execution restrictions through memory-safe runtime architectures and controlled host interaction mechanisms. While containers offer process-level isolation, WebAssembly introduces an additional security boundary that further limits potential attack vectors.

The analysis indicates that WebAssembly-based execution environments achieve a more favorable balance between security and performance compared with traditional container-only approaches.

4.4.2 Virtual Machines versus WebAssembly-Based Isolation

Virtual machines have historically been used to isolate untrusted workloads due to their strong separation from host systems. However, they often introduce significant computational overhead and slower deployment times.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

The evaluation demonstrates that WebAssembly-based environments provide lightweight isolation capabilities while preserving many of the security advantages associated with traditional virtualization. Faster startup times and reduced resource consumption make WebAssembly particularly attractive for dynamic execution scenarios. Consequently, WebAssembly presents a viable alternative for environments requiring both security and operational efficiency.

4.4.3 Existing Security Approaches versus the Proposed Framework

Most existing security approaches focus on individual technologies such as container hardening, runtime monitoring, or orchestration security. While these solutions address specific aspects of execution security, they frequently operate in isolation.

The proposed framework differs by integrating multiple security mechanisms into a unified architecture. This defense-in-depth approach combines policy enforcement, runtime isolation, orchestration controls, monitoring capabilities, and workload containment strategies within a single operational model.

The analysis suggests that this integrated approach provides broader protection against execution-related threats than solutions focused on individual security controls.

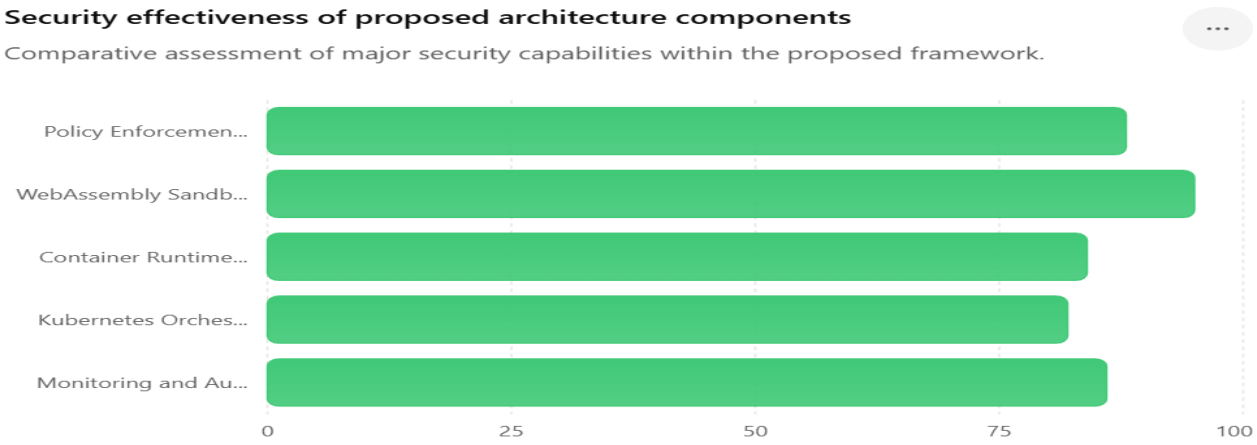
4.5 Summary of Findings

The results demonstrate that the proposed architecture provides a secure and scalable framework for autonomous workload execution within distributed cloud-native environments. The integration of WebAssembly sandboxing, container runtime isolation, Kubernetes orchestration, and policy-based controls creates multiple layers of defense that collectively reduce attack surfaces and improve operational resilience.

Key findings from the evaluation include:

- ✓ Enhanced execution isolation through combined WebAssembly and container-based sandboxing.
- ✓ Effective enforcement of least-privilege access controls.
- ✓ Improved containment of execution-related security threats.
- ✓ Reduced attack surfaces through layered security mechanisms.
- ✓ Efficient workload deployment with minimal runtime overhead.
- ✓ Strong scalability and orchestration capabilities through Kubernetes.
- ✓ Comprehensive monitoring and auditability across execution lifecycles.

Overall, the findings indicate that the proposed framework provides a practical and vendor-neutral foundation for securing autonomous execution environments while maintaining the flexibility and scalability required by modern cloud-native infrastructures.



5. DISCUSSION

5.1 Interpretation of Findings

The results demonstrate that secure execution remains one of the most critical challenges facing autonomous computational systems. As software agents increasingly perform tasks that involve code generation, workflow automation, resource allocation, and interaction with external systems, traditional security mechanisms are often insufficient to manage the associated risks. The findings indicate that a layered security architecture combining WebAssembly sandboxing, container runtime isolation, Kubernetes orchestration, and policy-based controls can significantly improve the security posture of distributed execution environments.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

The proposed framework addresses a fundamental limitation observed in many existing approaches, namely the reliance on a single security mechanism to protect complex workloads. Rather than depending exclusively on containers, access control policies, or orchestration platforms, the architecture incorporates multiple protective layers that work together to reduce attack surfaces and improve resilience against security breaches. This layered approach aligns with contemporary defense-in-depth principles and supports the secure execution of dynamically generated workloads within cloud-native environments.

The evaluation further suggests that isolation mechanisms play a central role in mitigating execution-related threats. By restricting workload access to system resources and enforcing strict runtime boundaries, the architecture minimizes opportunities for unauthorized interactions and limits the potential impact of compromised workloads. These findings reinforce the importance of secure execution environments as a foundational component of autonomous computing infrastructures.

5.2 Security Benefits of the Proposed Architecture

5.2.1 Reduced Attack Surface

One of the most significant advantages of the proposed framework is its ability to reduce the attack surface associated with workload execution. Traditional execution environments often expose broad sets of system resources, interfaces, and operating system functionalities to running applications. Such exposure increases opportunities for attackers to exploit vulnerabilities and gain unauthorized access.

The integration of WebAssembly runtimes introduces stricter execution boundaries that limit direct interaction with host resources. Combined with policy enforcement and container isolation, these restrictions reduce the number of accessible attack vectors available to malicious actors. Consequently, the architecture provides stronger protection against unauthorized operations and execution-based attacks.

Furthermore, the implementation of least-privilege principles ensures that workloads receive only the permissions required to perform their intended functions. This approach minimizes unnecessary access rights and decreases the likelihood of privilege escalation incidents.

5.2.2 Stronger Isolation Mechanisms

The findings indicate that the combination of WebAssembly sandboxing and container technologies establishes multiple isolation boundaries that improve overall execution security. Traditional container environments rely primarily on operating system-level isolation mechanisms, which may be affected by kernel vulnerabilities or misconfigurations. In contrast, WebAssembly introduces an additional layer of protection through memory-safe execution environments and controlled host interactions.

The dual-isolation model enhances workload containment by ensuring that even if one security layer is compromised, additional safeguards remain in place. Such redundancy is particularly important in environments where dynamically generated actions may exhibit unpredictable behavior.

Strong isolation also contributes to operational stability by preventing workloads from interfering with neighboring services or shared infrastructure resources. This capability is essential for maintaining reliability within distributed cloud-native systems.

5.2.3 Enhanced Monitoring and Accountability

The Monitoring and Audit Layer provides comprehensive visibility into execution activities across the architecture. Continuous monitoring enables the detection of abnormal behavior, policy violations, and potential security incidents before they escalate into significant threats.

Comprehensive logging and audit capabilities further support forensic investigations, compliance requirements, and incident response activities. By maintaining detailed records of workload behavior and execution events, organizations can improve accountability and strengthen governance over autonomous operations.

The integration of monitoring mechanisms with orchestration platforms also facilitates real-time security assessment, enabling administrators to respond rapidly to emerging threats.

5.3 Practical Implications

5.3.1 Enterprise Automation Systems

Organizations increasingly rely on autonomous systems to automate business processes, data analysis, customer support operations, and software development activities. The proposed architecture offers a secure foundation for deploying such systems within enterprise environments while maintaining compliance with security and governance requirements.

By isolating execution environments and enforcing policy-driven controls, enterprises can reduce risks associated with automated workflows and improve confidence in autonomous operational processes.

5.3.2 Software Engineering and Development Operations

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Modern software engineering practices increasingly incorporate automated tools capable of generating, testing, and deploying code. Secure execution environments are essential for preventing generated code from introducing vulnerabilities or compromising development infrastructure.

The proposed framework provides a controlled environment for executing generated scripts, testing software components, and validating deployment processes. Such capabilities support safer automation within continuous integration and continuous deployment (CI/CD) pipelines.

5.3.3 Distributed Cloud-Native Applications

Cloud-native systems require scalable and resilient execution platforms capable of managing large numbers of workloads across distributed infrastructures. Kubernetes orchestration enables efficient resource management and workload scheduling, while WebAssembly sandboxing strengthens execution security.

The combination of these technologies creates a practical solution for organizations seeking to deploy secure distributed applications without sacrificing scalability or performance.

5.3.4 Critical Infrastructure and High-Security Environments

Industries such as finance, healthcare, telecommunications, and government services often operate under stringent security requirements. The proposed architecture offers several features that align with these requirements, including workload isolation, access control enforcement, continuous monitoring, and auditability.

These capabilities may support the adoption of autonomous execution technologies in environments where security and reliability are paramount concerns.

5.4 Relationship to Existing Literature

The findings of this study are consistent with previous research emphasizing the importance of isolation and sandboxing mechanisms in secure execution environments. Prior studies have demonstrated the effectiveness of container sandboxing and runtime restrictions in reducing attack surfaces and mitigating security risks. The present study extends this body of knowledge by integrating these concepts with WebAssembly-based execution environments and distributed orchestration platforms.

Research on WebAssembly security has highlighted its advantages in memory safety, controlled execution, and platform independence. The results obtained in this study support these observations and further suggest that WebAssembly can serve as a foundational security component within modern cloud-native architectures.

Similarly, existing literature on Kubernetes security emphasizes the need for robust access controls, monitoring systems, and orchestration safeguards. The proposed framework incorporates these recommendations and demonstrates how orchestration security can be integrated with execution isolation mechanisms to establish a more comprehensive security model.

Overall, the study contributes to the growing body of research focused on secure cloud-native computing and distributed execution environments by proposing an integrated approach that combines multiple security technologies within a unified architecture.

5.5 Limitations of the Proposed Framework

Despite the advantages identified during the evaluation, several limitations should be acknowledged.

5.5.1 Dependence on Runtime Security

The effectiveness of the proposed framework depends heavily on the security of underlying runtime environments. Vulnerabilities within WebAssembly runtimes, container technologies, or orchestration components may reduce the effectiveness of isolation mechanisms and create opportunities for exploitation.

5.5.2 Configuration Complexity

The integration of multiple security layers introduces operational complexity. Organizations implementing the proposed framework must properly configure access controls, monitoring systems, policy engines, and orchestration settings to achieve desired security outcomes.

Misconfigurations may weaken security protections and increase exposure to potential threats.

5.5.3 Limited Empirical Deployment Evaluation

This study focuses primarily on architectural design and conceptual security analysis. Although the framework demonstrates strong theoretical security characteristics, large-scale empirical deployment studies would provide additional insights regarding performance, scalability, and operational effectiveness under real-world conditions. Future implementations should therefore include experimental evaluations within production-grade environments.

5.6 Future Research Directions

Several opportunities exist for extending this research and enhancing secure execution architectures.

5.6.1 Integration with Confidential Computing

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Future research may explore the integration of confidential computing technologies such as trusted execution environments (TEEs) and hardware-based security mechanisms. These technologies could provide additional protection for sensitive workloads and data during execution.

5.6.2 Adaptive Security Policy Engines

Static security policies may not adequately address the dynamic nature of autonomous systems. Future studies could investigate adaptive policy engines capable of adjusting security controls in response to changing operational conditions and threat landscapes.

5.6.3 Decentralized Security Architectures

Emerging distributed computing models may benefit from decentralized security frameworks that eliminate single points of failure and improve resilience. Research into blockchain-supported audit systems and decentralized trust mechanisms may offer valuable enhancements to future execution architectures.

5.6.4 Large-Scale Performance Validation

Further studies should conduct extensive performance evaluations across diverse deployment environments to assess scalability, resource utilization, latency, and security effectiveness under varying workload conditions. Such evaluations would strengthen understanding of the practical feasibility of the proposed architecture.

5.7 Discussion Summary

The discussion highlights the effectiveness of combining WebAssembly sandboxing, container runtime isolation, Kubernetes orchestration, and policy-driven controls within a unified security architecture. The findings suggest that the proposed framework significantly improves workload isolation, reduces attack surfaces, strengthens execution governance, and supports scalable deployment within distributed cloud-native environments. While implementation challenges and research gaps remain, the architecture provides a promising foundation for secure autonomous execution systems and offers valuable directions for future advancements in cloud-native security.

6. CONCLUSION

6.1 Summary of the Study

The increasing adoption of autonomous computational systems has transformed the way digital tasks are executed across modern computing environments. As these systems become capable of generating, planning, and executing actions with minimal human intervention, concerns regarding execution security, resource protection, and operational governance have become increasingly significant. The ability to execute dynamically generated actions introduces new attack vectors that traditional security mechanisms are often not designed to address effectively. Consequently, there is a growing need for execution environments that can provide strong isolation, controlled resource access, continuous monitoring, and scalable workload management.

This study addressed these challenges by proposing a distributed and sandboxed execution architecture that integrates WebAssembly runtimes, container technologies, Kubernetes orchestration, policy enforcement mechanisms, and monitoring capabilities within a unified security framework. The architecture was designed using a defense-in-depth approach in which multiple security layers work together to reduce attack surfaces and improve resilience against execution-related threats.

The study reviewed existing literature on sandboxing technologies, container security, WebAssembly-based execution environments, and Kubernetes security mechanisms. The review identified significant advancements in each area while also revealing a lack of integrated frameworks specifically designed to secure autonomous execution environments. This gap provided the foundation for the development of the proposed architecture.

Through architectural analysis and security evaluation, the study demonstrated how layered security controls can improve workload isolation, restrict unauthorized access to resources, contain potentially harmful activities, and support secure execution within distributed cloud-native infrastructures. The findings suggest that combining WebAssembly sandboxing with Kubernetes orchestration offers a practical and scalable approach to strengthening execution security while maintaining operational efficiency.

6.2 Major Contributions

This study makes several contributions to the fields of cloud-native security, distributed computing, and secure execution architectures.

First, it presents a vendor-neutral security framework that combines multiple established technologies into a unified execution model. Unlike existing approaches that focus on individual security mechanisms, the proposed architecture integrates sandboxing, orchestration, monitoring, and policy enforcement within a comprehensive security ecosystem.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Second, the study demonstrates the value of WebAssembly as a secure execution technology for distributed workloads. By leveraging memory-safe execution environments and restricted host interactions, WebAssembly provides stronger workload isolation while maintaining lightweight operational characteristics.

Third, the research highlights the role of Kubernetes as a secure orchestration platform capable of supporting large-scale deployment, workload management, and policy enforcement across distributed environments. The integration of orchestration and execution security contributes to a more holistic approach to infrastructure protection.

Fourth, the proposed architecture establishes a foundation for implementing defense-in-depth principles within modern execution environments. The combination of policy validation, sandbox isolation, runtime monitoring, and orchestration controls creates multiple security barriers that collectively improve system resilience.

Finally, the study contributes to the growing body of knowledge concerning secure execution environments by identifying research gaps and proposing future directions for enhancing execution security in increasingly complex cloud-native ecosystems.

6.3 Practical Implications

The findings have practical relevance for organizations seeking to deploy secure execution environments within enterprise, industrial, and cloud computing contexts. The proposed framework may support secure automation initiatives by reducing risks associated with dynamically generated workloads while maintaining scalability and operational flexibility.

Organizations operating within highly regulated sectors such as finance, healthcare, telecommunications, and government services may particularly benefit from the framework's emphasis on workload isolation, monitoring, auditability, and policy-based governance. These capabilities align with many contemporary security and compliance requirements.

The architecture may also serve as a foundation for future development of secure distributed systems that require robust execution controls and efficient resource management.

6.4 Final Remarks

Secure execution environments are becoming increasingly important as computational systems continue to evolve toward greater autonomy and operational complexity. While significant progress has been made in areas such as container security, WebAssembly sandboxing, and cloud orchestration, securing dynamically generated workloads remains a complex challenge requiring coordinated security mechanisms across multiple architectural layers.

The framework proposed in this study demonstrates that combining WebAssembly-based sandboxing, container runtime isolation, Kubernetes orchestration, policy enforcement, and continuous monitoring can significantly strengthen execution security within distributed cloud-native environments. By integrating these technologies into a cohesive architecture, organizations can establish secure, scalable, and resilient execution infrastructures capable of supporting future advances in autonomous computing.

Although additional research is required to validate the framework under large-scale production conditions, the proposed architecture provides a promising foundation for addressing emerging security challenges in modern distributed systems. Future developments in confidential computing, adaptive security policies, decentralized trust models, and advanced monitoring technologies are expected to further enhance the effectiveness of secure execution environments and contribute to the continued evolution of trustworthy cloud-native infrastructures.

7. REFERENCES

1) Aljuhani, A., Alghamdi, M., & Alshammari, R. (2025). Shadowkube: Enhancing Kubernetes security with behavioral monitoring and honeypot integration. *Cybersecurity, 8*(1), Article 72. https://doi.org/10.1186/s42400-025-00372-7
2) Bazzani, M., Lamberti, F., Demartini, C., & Cannavò, A. (2025). WebAssembly and security: A review. *Computer Science Review, 55*, 100728. https://doi.org/10.1016/j.cosrev.2025.100728
3) Di Federico, G., Continella, A., & Zuddas, D. (2026). WASP: Stack protection for WebAssembly. *Journal of Systems Architecture, 172*, 103666. https://doi.org/10.1016/j.sysarc.2025.103666
4) Ghafari, M., Shen, Y., & Aryani, A. (2019). Practical and effective sandboxing for Linux containers. *Empirical Software Engineering, 24*(6), 4034–4070. https://doi.org/10.1007/s10664-019-09737-2
5) Hallak, B., Serhani, M. A., & Salah, K. (2024). SoK: Analysis techniques for WebAssembly. *Future Internet, 16*(3), 84. https://doi.org/10.3390/fi16030084

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

6) Kim, J., Park, H., & Lee, S. (2024). Optimus: Association-based dynamic system call filtering for container attack surface reduction. *Journal of Cloud Computing, 13*(1), Article 52. https://doi.org/10.1186/s13677-024-00639-3
7) Lehmann, D., Pradel, M., & Rossow, C. (2022). Wasmati: An efficient static vulnerability scanner for WebAssembly. *Computers & Security, 118*, 102745. https://doi.org/10.1016/j.cose.2022.102745
8) Sjösten, A., Hedin, D., & Sabelfeld, A. (2023). SecWasm: Information flow control for WebAssembly. In *Programming Languages and Systems* (pp. 95–119). Springer. https://doi.org/10.1007/978-3-031-22308-2_5
9) Watt, C., Schmitt, A., Mullen, E., Ren, L., Giannarakis, N., & Swamy, N. (2022). Provably-safe multilingual software sandboxing using WebAssembly. *Proceedings of the ACM on Programming Languages, 6*(POPL), 1–30. https://doi.org/10.1145/3498702
10) Zhang, Y., Chen, X., & Liu, H. (2025). Losing control: Exposing security weaknesses of Kubernetes control plane interfaces. *Computers & Security, 148*, 104165. https://doi.org/10.1016/j.cose.2025.104165