

Article

cyberSPADE: A Hierarchical Multi-Agent Architecture for Coordinated Cyberdefense

Lucía Alba Torres ^{1,*} , Miguel Rebollo ¹ , Javier Palanca ¹  and Mario Aragonés Lozano ² 

¹ VRAIN–Valencian Research Institute for Artificial Intelligence, Universitat Politècnica de València, Camí de Vera s/n, 46022 València, Spain; mrebollo@upv.es (M.R.); jpalanca@dsic.upv.es (J.P.)

² Communications Department, Universitat Politècnica de València, Camí de Vera s/n, 46022 València, Spain; maragones@teleco.upv.es

* Correspondence: lalbtor1@etsinf.upv.es;

Abstract

Modern cyber threats demand coordinated defensive strategies that extend beyond centralized security mechanisms. However, existing multi-agent platforms exhibit critical limitations in explicit communication and real-time coordination for cyberdefense operations. This work proposes a hierarchical multi-agent architecture for autonomous cyberdefense that addresses these limitations through structured inter-agent communication and distributed coordination. The architecture integrates a centralized monitor agent with specialized defensive swarms deployed across operational hosts. It is implemented using SPADE 4.1 (Smart Python Agent Development Environment) to enable XMPP-based (Extensible Messaging and Presence Protocol) communication with low-latency messaging and location transparency. Four specialized swarms—Network Defender, Host Defender, Anomaly Detection, and Forensic and Recovery—perform autonomous defensive tasks. A secure authentication mechanism ensures trusted communication between monitor and deployer agents. The system was evaluated in a controlled virtualized environment using the Network Defender Swarm as an illustrative case. The experimental results focus on internal coordination behavior, messaging efficiency, and end-to-end detection time across increasing levels of parallelism. A scan agent scalability analysis shows that moderate parallelism (2–16 agents) yields the lowest Total Detection Time (12.88 s across the full TCP port range), while excessive agent counts degrade performance. Results demonstrate how the proposed architecture supports low-latency communication, efficient coordination, and parallel task execution. Message latency benchmarks show improvements compared to classical agent frameworks such as JADE. These findings provide initial evidence that communication-centric multi-agent architectures can facilitate coordinated and adaptive cyberdefense operations, while serving as a platform for further experimental evaluation.

Keywords: multi-agent systems; cyberdefense; SPADE; distributed architecture; swarm intelligence; autonomous security; coordinated defense



Academic Editor: Giorgio Giacinto

Received: 20 December 2025

Revised: 31 January 2026

Accepted: 2 February 2026

Published: 8 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The increasing digitalization of critical infrastructures, public services, and enterprise environments has significantly increased the complexity of the cybersecurity landscape. Modern attacks are increasingly automated, coordinated, and adaptive, exceeding the capabilities of traditional defensive mechanisms, which are often centralized and dependent on static rule-based detection [1–3]. This gap has driven a reconceptualization of defensive

operations, shifting from isolated detection tasks toward multi-agent interaction problems in which multiple autonomous entities must cooperate under uncertainty and adversarial dynamics [4,5].

Multi-agent systems (MAS) have emerged as a promising paradigm for addressing these challenges by modelling cyber defence as a distributed network of autonomous agents capable of reasoning, coordinating, and allocating tasks effectively. Recent research shows that adversarial behaviors such as strategic exploration, adaptive privilege escalation, and coordinated disruption are more effectively understood and mitigated from distributed, cooperative perspectives [4–6]. However, an active debate persists on the optimal degree of coordination between agents. Although some approaches advocate purely decentralized architectures based solely on local observation, others argue that explicit communication is essential to maintain shared situational awareness and respond to sophisticated adversaries [7,8]. This tension highlights the need to investigate MAS architectures that support structured inter-agent communication without introducing additional vulnerabilities or bottlenecks [9].

Despite conceptual progress, current simulation platforms and cyber-range environments (including CybORG, MITRE Caldera, NS-3, and Mininet) exhibit significant limitations when evaluating communicative MAS at the agent-abstraction level [10–13]. Although valuable for cyberdefense experimentation, these platforms rarely provide native mechanisms for semantic messaging, complex coordination, or real-time intelligence sharing, all of which are fundamental for cooperative MAS [10,14,15]. As a result, they offer limited support for the systematic experimental evaluation of inter-agent communication and coordination overhead in cooperative cyberdefense systems [16,17].

This work addresses these limitations by proposing a distributed MAS architecture and experimental testbed for evaluating explicit communication and coordination strategies in autonomous cyberdefense systems. The proposed system integrates a centralized strategic layer, embodied by a monitor agent, with a decentralized operational layer composed of specialized swarms deployed across multiple hosts. The architecture employs the SPADE platform to enable structured communication, leveraging its support for XMPP to provide semantic messaging, concurrent behaviors, and transparent deployment in distributed environments [18–21]. While the system includes concrete defensive capabilities, the primary contribution of this work lies in the architecture and testbed itself, rather than in proposing new detection or analysis techniques. The initial experimental results are intended to validate the feasibility of the proposed architecture and to illustrate how the testbed enables quantitative evaluation of inter-agent communication efficiency under controlled conditions.

The remainder of the article is organized as follows. Section 2 reviews the state of the art in MAS for cybersecurity and communication-enabled agent frameworks. Section 3 introduces the SPADE framework and communication model underpinning the proposed system. Sections 4 and 5 present the design and structure of the proposed multi-agent architecture. Section 6 describes the experimental environment used to evaluate the performance of the Multi-agent system. Section 7 reports the experimental results obtained from the prototype implementation. Section 8 discusses the implications, advantages, and limitations of the proposed approach and outlines future research directions.

2. Background

Recent research in cyberdefense has increasingly framed security operations as multi-agent interaction problems, rather than as isolated detection or classification tasks [1,5,22]. This paradigm shift acknowledges the limitations of traditional, centralized defensive mechanisms [2,3], which often rely on signature-based detection or static rules and struggle

to adapt to novel and evolving threats [2,22]. In contrast, modern attacks are often distributed, coordinated, and easier to replicate, placing defenders at a disadvantage when using centralized systems [6,23].

Modern multi-agent systems are commonly defined as networks of two or more autonomous agents that maintain private information states, pursue their own objectives, and interact through explicit communication channels or shared environments [24–26]. This perspective has gained relevance in cybersecurity as emerging autonomous attackers and defenders exhibit behaviors, such as strategic exploration, adaptive privilege escalation, or coordinated disruption, that cannot be fully understood by analyzing individual agents in isolation [1,6]. The growing attention to multi-agent architectures in cybersecurity stems from their demonstrated capabilities in autonomy, scalability, fault tolerance, and automatic parallelism [5].

MAS in cybersecurity is often categorised by its behavioral paradigms [27]. Reactive agents respond immediately to environmental stimuli and perform tasks such as anomaly detection and host-level monitoring. Deliberative agents reason about plans and long-term objectives, enabling strategic threat mitigation and coordinated responses [22]. Swarm-based agents operate in large, distributed collectives where simple local rules produce emergent global behavior, offering scalable approaches to network-wide defense. Together, these MAS paradigms highlight the flexibility and adaptability of autonomous defensive agents.

Empirical studies indicate that agent-to-agent interaction introduces qualitatively new security challenges and opportunities, including covert collusion channels, coordinated attacks, and cascading failures across distributed infrastructures [24]. These phenomena expose the limitations of traditional, centralised defensive mechanisms and underscore the need for decentralised, communicative, and cooperative agents capable of maintaining shared situational awareness and responding jointly to adversarial threats. In this context, explicit inter-agent communication, task allocation, and distributed decision-making are essential mechanisms for effective autonomous cyberdefense.

Benchmark environments, such as the CAGE Challenge 2 [24,28], provide concrete evidence of the effectiveness of MAS in cybersecurity scenarios. In CAGE’s adversarial enterprise-network scenario, where a Blue defender must counter diverse Red attacker strategies, hierarchical reinforcement-learning agents consistently achieve the strongest defensive performance. However, the significant variability of policies observed both within and across algorithmic classes—ranging from targeted defensive removal to widespread decoy deployment—indicates that defenders must adapt to heterogeneous attacker behaviors, such as the exploratory “Meander” adversary or the goal-directed “B-line” strategy. These results confirm that DRL-based approaches are promising candidates for autonomous defense, while also revealing critical limitations related to computational complexity, generalization, and attacker-specific specialization [24].

Implementing coordinated multi-agent systems for cybersecurity requires robust middleware platforms that support distributed deployment, structured communication, and concurrent agent behavior. However, significant disparities exist among available frameworks regarding their communication capabilities, performance characteristics, and suitability for real-time cyberdefense scenarios. Understanding these differences is essential for selecting an appropriate foundation for autonomous defensive systems.

JADE (Java Agent DEvelopment Framework) has been one of the most widely adopted platforms in the MAS community since its introduction in the late 1990s [29]. JADE implements the FIPA (Foundation for Intelligent Physical Agents) standards and provides comprehensive support for agent lifecycle management, yellow-page directory services, and messaging through an agent communication language (ACL). Its architecture is built

around the concept of containers, which host agents and can be distributed across multiple machines. JADE's maturity and extensive documentation have made it a reference implementation for FIPA-compliant systems. However, JADE exhibits several limitations that become critical in high-performance or resource-constrained scenarios. Its reliance on heavyweight Java Virtual Machine instances introduces significant memory overhead, particularly when deploying large numbers of agents [30]. Additionally, JADE's synchronous message-handling model can create bottlenecks when agents must process high volumes of concurrent communications, a common requirement in real-time cyberdefense scenarios.

FIPA-OS, another FIPA-compliant platform developed by Nortel Networks and later maintained by Emorpha, was designed with a strong emphasis on standards compliance and interoperability [31]. FIPA-OS introduced innovations in agent mobility and provided early implementations of FIPA interaction protocols. However, the platform's development has largely stagnated, and its adoption in contemporary research has diminished significantly. Its architecture, while theoretically sound, lacks native support for modern asynchronous programming paradigms and does not integrate well with current communication technologies such as XMPP.

Zeus, developed at British Telecom, offered an alternative approach focused on providing high-level abstractions for agent coordination and collaborative problem-solving [32]. Zeus included visual development tools and emphasized role-based agent design, making it accessible for rapid prototyping and development. However, Zeus's proprietary communication mechanisms limited interoperability with other platforms, and its development was discontinued in the mid-2000s, rendering it unsuitable for modern deployments.

More recent platforms have attempted to address these limitations. SPADE (Smart Python Agent Development Environment) represents a significant departure from the Java-centric paradigm by adopting Python as its implementation language and XMPP (Extensible Messaging and Presence Protocol) as its communication substrate [33]. This architectural decision enables SPADE agents to leverage Python's extensive ecosystem for data analysis, machine learning, and network security tools while benefiting from XMPP's mature infrastructure for real-time messaging. SPADE's asynchronous execution model, built on Python's *asyncio* library, fundamentally differs from JADE's threading approach, allowing agents to handle concurrent behaviors with significantly lower overhead. Furthermore, SPADE's integration with XMPP provides native support for presence notifications, enabling agents to maintain awareness of their peers' availability without requiring centralized directory services.

Other notable platforms include JASON [34], which focuses on agent programming using the AgentSpeak language and is particularly suited for belief–desire–intention (BDI) architectures. NetLogo, on the other hand, excels in agent-based modeling and simulation but lacks the communication infrastructure required for operational cyberdefense deployments [35]. Jadex combines BDI reasoning with FIPA compliance and offers active component support, but shares JADE's performance characteristics due to its Java foundation [36].

Despite the growing adoption of multi-agent approaches in cybersecurity, implementing explicit communication mechanisms between agents in simulation platforms and cyber-training environments remains a significant challenge. Environments such as CybORG, which are fundamental for research on autonomous cyber-operations agents [28], focus primarily on the development of decision-making agents but exhibit notable gaps in supporting inter-agent communication. Recent work has highlighted the need to assess how large language models perform in multi-agent scenarios when interacting with other autonomous cyberdefense (ACD) agents [10]. Indeed, current research proposes the creation of new communication protocols explicitly designed for heterogeneous ACD

agents, including reinforcement learning agents and LLMs, within CybORG to enable effective coordination [10].

The absence of standardized, explicit communication protocols on these platforms fundamentally limits agents' ability to coordinate complex actions, share real-time intelligence, and collaborate effectively against sophisticated cyber threats [13]. This limitation is not merely operational but architectural: platforms designed around individual agent decision-making often lack the semantic messaging layers, presence mechanisms, and structured interaction protocols necessary for sophisticated multi-agent coordination. Without these capabilities, agents cannot maintain shared situational awareness, negotiate task allocation, or implement distributed consensus mechanisms, all of which are essential for adaptive cyberdefense.

The validation and training of multi-agent cyberdefense systems require robust, realistic test environments. Cyber range and simulation platforms such as NS-3, CORE, Mininet, and GNS3 are widely used tools for this purpose [11,12]. NS-3 and Mininet stand out as accessible options for large-scale network experimentation in simulation and emulation, respectively [37]. However, while simulators and emulators offer controlled environments for reproducible experimentation, they often lack realism [11].

Some of these tools support agent deployment and collaboration through information exchange. For example, NS-3 can be used for multi-threaded agents that collaborate by exchanging control signals [38], or even integrated with multi-LLM agent frameworks to generate simulations [39]. Nevertheless, they typically do not natively incorporate the complexity of communicative multi-agent systems at an agent-level abstraction. Integrating complex communicative multi-agent systems into these platforms often requires additional frameworks or custom implementations, underscoring a limitation in their intrinsic support for advanced multi-agent interaction [14,15].

Although these platforms are excellent at network-level simulation and provide valuable environments for testing individual agents' decision-making capabilities, they lack the native abstractions necessary to model the rich communication patterns, coordination protocols, and emergent behaviors that characterize truly cooperative multi-agent systems [16,17]. This gap is particularly significant for cyberdefense research, where evaluating collective coordination, information sharing patterns, and system-wide resilience—not just individual agent performance—is essential.

Tables 1 compare representative cybersecurity benchmarks and cyber range platforms with the proposed multi-agent system, cyberSPADE, across dimensions of agent communication, coordination, heterogeneity, and deployment orientation.

Table 1. Comparison of Communication, Coordination, and Deployment Features of Cybersecurity Platforms.

Platform	Agent Comm.	Multi-Agent Coord.	Comm. Security	Heterog. Agents	Deployment Orientation
CAGE Challenge 2	Limited	Centralized	X	Limited	Evaluation
CybORG	X	Limited	X	Partial	Evaluation
NS-3	Ad hoc	X	X	X	Simulation
Mininet	Ad hoc	X	X	X	Simulation
CORE / GNS3	Ad hoc	X	X	X	Simulation
cyberSPADE	✓	✓	✓	✓	Operational

Agent Communication: Native support for explicit inter-agent messaging. Multi-Agent Coordination: Support for decentralized and cooperative behaviors. Communication Security: Built-in security mechanisms for agent communication. Heterogeneous Agents: Support for diverse agent types and capabilities. Deployment Orientation: Designed for operational deployment rather than isolated evaluation. Ad hoc: Non-native, non-standard mechanisms implemented specifically for a given scenario. ✓: Feature supported. X: Feature not supported.

This gap underscores the need for a multi-agent architecture that supports both the conceptualization and practical implementation of coordination through rich inter-agent communication and complex interaction. Such an architecture must address three critical dimensions simultaneously: (1) provide explicit, structured communication mechanisms that enable sophisticated coordination protocols, (2) achieve the performance characteristics necessary for real-time threat response, and (3) incorporate security-by-design principles that protect the communication infrastructure itself from adversarial compromise. By addressing these dimensions, the proposed architecture aims to bridge the gap between theoretical multi-agent coordination models and practical autonomous cyberdefense systems, thereby enabling coordinated and adaptive responsiveness to emerging cyber threats.

Table 2 summarizes the key characteristics of the evaluated multi-agent platforms. The comparison highlights that while several frameworks provide FIPA compliance and distributed deployment capabilities, SPADE uniquely combines asynchronous execution, native presence mechanisms, and extensive integration with the Python cybersecurity ecosystem. These features make it particularly well-suited for implementing the coordinated, real-time defensive architectures required in modern cyberdefense scenarios.

Table 2. Comparison of Multi-Agent System Platforms.

Platform	FIPA Compliant	Async Model	Low Latency	Presence Support	Active Dev	Language	Security Integration
JADE	✓	X	X	X	✓	Java 8	Limited
FIPA-OS	✓	X	X	X	X	Java 8	Limited
Zeus	X	X	X	X	X	Java 8	Limited
SPADE	✓	✓	✓	✓	✓	Python 3.12	Extensive
JASON	✓	X	X	X	✓	Java 8	Limited
Jadex	✓	X	X	X	✓	Java 8	Limited
NetLogo	X	X	N/A	X	✓	Java	Minimal

Low Latency: Message processing < 1 ms; Presence Support: XMPP-like presence notifications; Active Dev: Currently maintained and updated; Security Integration: Native support for cybersecurity libraries. Legend: ✓: Feature supported. X: Feature not supported.

3. SPADE Platform Architecture and Capabilities

The SPADE platform provides the foundational infrastructure for the multi-agent architecture proposed in this work. This section presents an analysis of SPADE's design principles, core components, and technical features that directly support the requirements of autonomous cyberdefense systems.

3.1. Platform Overview

SPADE represents a modern approach to multi-agent system development, built on the principle that agent platforms should leverage existing, mature communication protocols rather than implementing proprietary messaging layers [33]. The platform is implemented entirely in Python 3, enabling seamless integration with the extensive ecosystem of security, machine learning, and data analysis libraries that have become standard in cybersecurity research and operations. Unlike classical agent platforms that treat communication as an internal implementation detail, SPADE adopts XMPP (Extensible Messaging and Presence Protocol) as its native communication substrate, providing agents with access to a battle-tested, standardized messaging infrastructure used by millions of users worldwide. The platform's architecture follows a minimalist philosophy: rather than prescribing rigid agent structures or enforcing specific reasoning models, SPADE provides a flexible foundation that enables developers to implement diverse agent paradigms—reactive, deliberative, or hybrid—using a consistent set of primitives. This flexibility is critical in

cyberdefense contexts where different defensive functions may require fundamentally different operational logics.

3.2. XMPP Integration and Communication Infrastructure

SPADE's reliance on XMPP provides several advantages for multi-agent cyberdefense systems [40]. Each SPADE agent is assigned a unique Jabber ID (JID) of the form `username@domain/resource`, which serves as both its identifier and its communication endpoint. This identifier is globally unique and location-independent, meaning that agents can be restarted or redeployed on different hosts while preserving their communication identity [18]. The XMPP presence mechanism provides a native solution for agent discovery and availability monitoring. Agents can subscribe to the presence of other agents and receive automatic notifications when those agents come online, go offline, or change their status. This capability eliminates the need for centralized yellow-page services or periodic polling mechanisms. As a result, it reduces network overhead and implementation complexity. In the context of cyberdefense, presence notifications enable the monitor agent to maintain real-time awareness of which defensive agents are currently operational without requiring explicit heartbeat messages. SPADE can operate with external XMPP servers such as ejabberd or Prosody, enabling agents to communicate across organizational boundaries when necessary, or with its embedded lightweight XMPP server (Pijabber) for isolated deployments.

3.3. Behavior Model

The core abstraction in SPADE is the behavior, which represents a specific task or routine performed by an agent. SPADE provides several predefined behavior types that correspond to common agent patterns:

- Cyclic behaviors execute continuously in a loop, processing incoming messages or performing periodic checks. These are suitable for monitoring tasks, such as continuously watching for new vulnerability reports.
- One-shot behaviors execute exactly once and then terminate. They are used for initialization tasks, single-event responses, or trigger-based actions.
- Periodic behaviors execute at fixed time intervals, providing scheduled execution without requiring explicit timing logic. This behavior type is ideal for establishing baselines or conducting periodic integrity checks.
- Timeout behaviors execute once after a specified delay, enabling deferred actions or timeout-based fallback mechanisms.
- Finite State Machine (FSM) behaviors allow agents to implement complex, state-based logic where transitions between states are triggered by messages or conditions. FSM behaviors are beneficial for implementing multi-phase protocols, such as the secure onboarding sequence between monitor and deployer agents.

3.4. Message Templates and Semantic Communication

SPADE provides a flexible, template-based system that enables agents to define which message characteristics they are interested in precisely and to disregard irrelevant fields. A message template specifies matching criteria for incoming messages, such as sender identity, performative type (e.g., INFORM, REQUEST, QUERY), conversation identifier, or message ontology.

SPADE messages can be enriched with custom metadata fields carried within the XMPP message envelope, allowing agents to embed structured data, authentication tokens, or cryptographic signatures within the message. Message bodies can contain plain text, JSON-serialized data structures, or binary content encoded as Base64, providing flexibility

in how information is represented and transmitted. The platform automatically handles message serialization, routing, and delivery, abstracting away the complexities of XMPP stanza construction. At the same time, it still provides access to low-level protocol features when necessary. This layered approach enables rapid development of standard agent interactions while preserving the ability to implement custom protocols when required by security or performance constraints.

3.5. Concurrent Execution and Asynchronous Operations

One of SPADE's most significant architectural advantages is its deep integration with Python's `asyncio` library, which provides cooperative multitasking within a single event loop. Unlike traditional threading models, where the operating system manages context switches that may occur at arbitrary points during execution, `asyncio` allows agents to yield control at well-defined points in the program explicitly. This execution model significantly reduces classes of concurrency bugs related to race conditions and lock contention. For cyberdefense agents that must perform I/O-bound operations—such as querying remote vulnerability databases, scanning network ports, or reading system logs—`asyncio`'s non-blocking design provides substantial performance benefits. An agent can initiate hundreds or thousands of concurrent network operations without allocating a thread per connection, enabling efficient resource utilization even on systems with limited memory or CPU capacity.

3.6. Distributed Deployment and Location Transparency

SPADE agents can be deployed on separate physical machines, virtual machines, or containers without requiring changes to agent code and with only minimal configuration, such as specifying the XMPP server address. Once connected to the XMPP server, agents can discover and communicate with each other regardless of their physical location, network topology, or IP address assignment. This location transparency is a direct consequence of XMPP's federated architecture and SPADE's use of JIDs as stable identifiers [21]. For the cyberdefense architecture proposed in this work, location transparency enables the monitor agent to coordinate with deployer agents across distributed hosts without maintaining explicit IP address mappings or routing tables. When a deployer agent connects to the XMPP server, it becomes automatically reachable by the monitor via its JID. If the deployer's host is migrated to a different network segment or assigned a new IP address, communication is transparently resumed, as the XMPP server routes messages based on JIDs rather than IP addresses.

This capability also simplifies the implementation of fault-tolerant mechanisms. When the monitor agent fails and is restarted on a different host, it reconnects to the XMPP server using the same JID and resumes communicating with all deployer agents. From the deployers' perspective, the monitor has simply transitioned from an offline to an online state, as detected via XMPP presence notifications, without requiring explicit reconfiguration or service discovery.

3.7. Platform Limitations and Considerations

While SPADE provides substantial advantages for implementing communicative MAS in cyberdefense, certain limitations and design considerations must be acknowledged. The platform's reliance on Python introduces runtime overhead compared to compiled languages. However, this penalty is generally negligible for I/O-bound security operations, where network latency typically dominates execution time. For computationally intensive tasks such as cryptographic operations or large-scale log parsing, agents can delegate work to compiled native libraries or external services. SPADE's XMPP dependency requires maintaining an XMPP server infrastructure, which introduces an additional component

that must be secured, monitored, and maintained. However, this trade-off is justified by the resulting communication capabilities and the maturity of available XMPP server implementations [41]. The embedded Pijabber server mitigates this concern for small-scale or isolated deployments. The platform's asynchronous execution model, while powerful, requires developers to adopt coroutine-based programming patterns that may differ from traditional imperative or threaded approaches. However, this learning curve is offset by the elimination of complex locking mechanisms and the improved reasoning about concurrent execution that *asyncio* provides. Despite these considerations, SPADE's combination of low-latency messaging, flexible agent architectures, support for distributed deployment, and extensive integration capabilities makes it uniquely suited to implementing the hierarchical, communicative multi-agent system proposed in this work.

4. System Architecture

The proposed system is based on a hierarchical yet distributed MAS architecture, referred to as *cyberSPADE*. This architecture enables coordinated cyberdefense across multiple hosts while preserving the autonomy required for rapid, context-sensitive reactions. Unlike traditional detection or monitoring frameworks, which are often monolithic, rule-centric, and limited in their capacity for explicit coordination, this architecture adopts a cooperative organisational model. It emphasises explicit message exchange, task specialization, and distributed deployment across networked hosts. Such a design aligns with well-established MAS principles, including division of labour, agent autonomy, and cooperative problem-solving [26,42].

At the conceptual level, the system integrates three architectural pillars. First, a centralised strategic layer is embodied by the monitor agent (deliberative agent), which resides on a supervisor host and maintains a system-wide perspective of the defended environment. Its responsibilities include consolidating observations produced by distributed agents, resolving inconsistencies, orchestrating defensive phases, and managing the lifecycle of all subordinate agents. Its role mirrors hierarchical coordination patterns widely used in cooperative MAS for cyberdefense and network management [43,44].

Second, a decentralized operational layer, deployed across multiple hosts, consists of lightweight, specialized agents organized into swarms (reactive agents). These worker agents perform focused defensive tasks, ranging from host-level service enumeration to internal integrity verification and anomaly detection, without requiring continuous supervision. Within a host, agents may communicate and coordinate with one another, even across different swarms, to enhance response effectiveness when collaboration is required. However, inter-host communication is restricted, preserving local autonomy and enforcing the trust boundaries established by the supervisor-host monitor.

Third, both layers are bound together by a standardised communication substrate implemented through the SPADE platform, which supports semantic messaging and interoperability. This design enables an apparent decoupling between agent roles. Figure 1 provides an overview of the system deployment, showing the supervisor host, operational hosts, and their interaction through the XMPP communication server.

The interaction between the strategic and operational layers produces a two-tier defensive model. While worker agents focus on fast, host-local reactions, the monitor agent provides vertical reasoning that integrates local perspectives into a coherent understanding of the global environment. This hybrid organisation enables rapid detection and initial assessment at the edges, coupled with informed, coordinated decision-making at the centre.

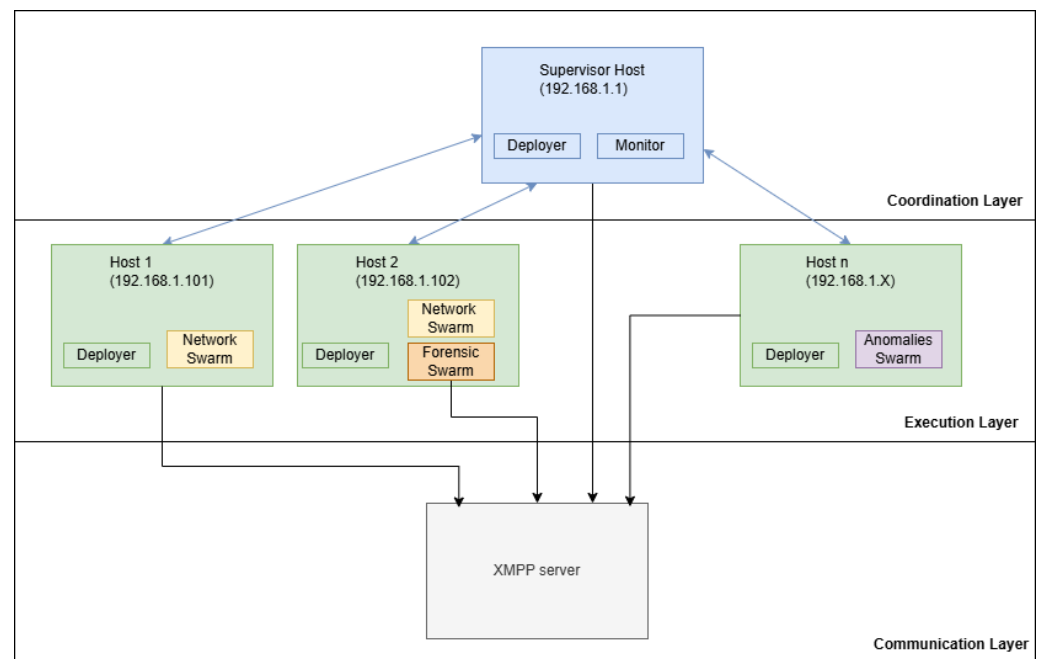


Figure 1. Architecture Diagram. The system architecture is organised into three layers: the Coordination Layer, which includes the monitor agent (blue); the Execution Layer, consisting of defensive swarms of specialised agents (green); and the Communication Layer, implemented via an XMPP message server (grey).

4.1. SPADE-Based Distributed Deployment Model

SPADE serves as the multi-agent middleware responsible for agent hosting, communication management, and behavior scheduling across the distributed system [33]. Its XMPP-based communication substrate provides a robust foundation for the coordination-centric multi-agent architectures, making it suitable for autonomous cyberdefense scenarios that require explicit inter-agent interaction.

The deployment architecture distinguishes between two node types. The supervisor host executes only the monitor agent and therefore concentrates all strategic reasoning. This host maintains a system-wide logical view of the defended environment and is responsible for all high-level reasoning. Each operational host, in contrast, runs a dedicated deployer agent that acts as the local authority responsible for instantiating Worker Agents. This separation creates a clear organisational boundary: while the monitor determines which actions are needed, deployers ensure those actions are correctly executed within their respective hosts.

Once the monitor identifies the need for a particular defensive capability, it initiates a deployment sequence by sending a structured request to the relevant deployer, as shown in Figure 2. Upon receiving this request, the deployer instantiates the required swarm. During operation, worker agents may exchange information directly within a host, for instance, to correlate vulnerability findings with anomalous behaviors. This exchange allows local cooperative behavior to emerge while reducing the need for continuous monitoring of low-level details. Their findings are consolidated by a designated reporter agent, which produces semantically normalised summaries that the monitor incorporates into its global state model.

This deployment model offers several advantages: it limits the monitor's exposure to low-level host operations, supports the enforcement of trust boundaries, and enables the instantiation of specialised swarms on demand without interrupting ongoing system activity.

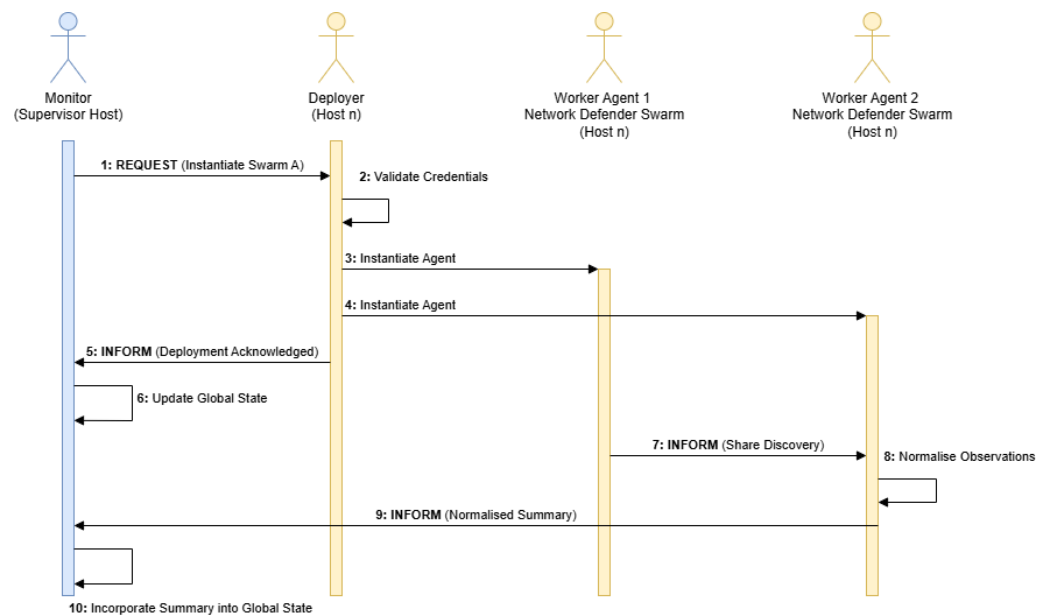


Figure 2. Deployment sequence diagram for on-demand swarm instantiation. The deployer validates the monitor’s credentials and manages the deployment request by instantiating the required worker agents. Worker agents cooperate to produce a normalised summary, which is reported back to the monitor and incorporated into the global system state. Colour coding highlights the separation between coordination activities (monitor, shown in blue), local execution activities (deployer and worker agents, shown in yellow), and reporting flows.

4.2. Secure Monitor–Deployer Communication

Given that deployer agents execute privileged actions on behalf of the monitor, the communication channel between them requires strong guarantees of authenticity, confidentiality, and resistance to adversarial interference. To this end, the architecture implements a multi-stage secure communication protocol that begins with a one-shot authentication mechanism based on a time-based one-time password, as defined in RFC 6238 [45].

When a new deployer is to be incorporated into the system, the operator issues a dedicated command on the monitor (e.g., “–register-deployer”). As a result, the monitor generates a TOTP shared secret and an initial provisioning token associated with that specific deployer instance. Both values are then delivered to the deployer through an out-of-band secure channel, such as manual provisioning by an operator or transfer via a physically controlled medium. The deployer also obtains the monitor’s public key during this provisioning phase, which is required to protect its first authenticated message.

During the initial handshake, as illustrated in Figure 3, the deployer computes a TOTP value from the shared secret. It sends an onboarding message encrypted under the monitor’s public key to the monitor. This message contains the provisioning token assigned during registration, as well as a freshly generated symmetric secret that will be used for all subsequent communication. Because the provisioning token and the TOTP code are valid only within a short time window and cannot be reused, this mechanism significantly reduces the risk of boot-time replay and impersonation attempts.

Once the initial authentication is complete, the monitor and deployer may optionally perform a post-quantum key establishment and authentication [46,47]. This step derives a symmetric session key that is designed to provide resistance against quantum-capable adversaries. If this step is performed, the derived secret replaces the initial session key; otherwise, the deployer continues using the symmetric secret generated during bootstrap. In both cases, all subsequent communications are encrypted using this symmetric key, enabling low-latency messaging without repeated use of costly asymmetric operations.

From that point onward, the monitor interacts with each deployer through a persistent, mutually authenticated secure channel. All deployment requests are cryptographically verified before execution, and deployers reject any message that does not originate from a valid monitor entity. This secure channel constitutes the trust backbone of the architecture, significantly reducing the risk that defensive operations are manipulated by adversaries seeking to deploy malicious agents or disrupt legitimate ones.

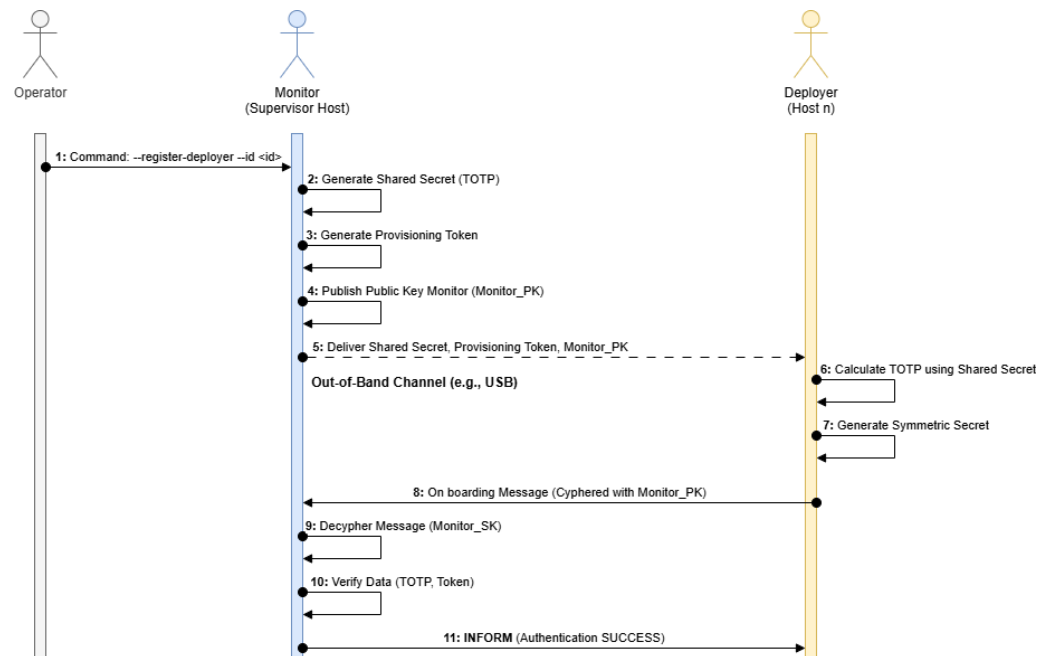


Figure 3. Deployment authentication sequence diagram. The protocol combines out-of-band provisioning, TOTP-based one-time authentication, and asymmetric cryptography to establish an initial trust relationship. Following successful verification, a symmetric secret is negotiated for low-latency encrypted communication. Colour coding distinguishes the roles of the operator (grey), the monitor running on the supervisor host (blue), and the deployer on the target host (yellow). Dashed arrows indicate out-of-band communication channels.

4.3. Fault Tolerance and Monitor Failover Mechanism

To ensure continuity of situational awareness in the event of a supervisor-host failure, the architecture incorporates a failover mechanism. All deployer agents periodically verify the monitor's availability. If the monitor becomes unreachable, deployers initiate a supervisor-side election procedure in which one of them is temporarily assigned the standby monitor role. This temporary monitor operates in a strictly passive mode: it collects and stores incoming reports from all swarms but does not issue deployment commands or orchestrate defensive phases. Its sole function is to maintain an updated situational snapshot until the original monitor becomes available again. Once the genuine monitor is restored, the temporary monitor securely transfers all accumulated reports and relinquishes its role, supporting continuity without compromising the architectural principle that only the supervisor-host monitor may perform strategic control.

In addition to this failover capability, the architecture also supports complete monitor migration, allowing the monitor to be relaunched on a different host—even one with a different IP address—without disrupting trust relationships. It is achieved by binding the monitor's identity to a persistent cryptographic keypair rather than to any physical network attribute. When the original monitor fails and is later reinstantiated on a new host, it announces its return to all deployers using the same cryptographic identity. The deployers verify this identity, re-establish their secure channels, and transmit the reports accumulated during the outage. Because trust is anchored to the monitor's persistent key

rather than to its location, the operating system, host, or IP address becomes irrelevant to the recovery process. This design enables monitor migration while preserving global consistency and ensuring that only the legitimate monitor can resume strategic control.

5. Agent Roles and Behaviors

The system's defensive capabilities emerge from the coordinated operation of multiple agent categories. Each agent type embodies a well-defined operational role, communication pattern, and behavioral model. While the monitor agent centralises strategic reasoning and the deployer agents instantiate the necessary components on each host, the majority of the defensive activity is carried out by specialised swarms of worker agents. These swarms implement distinct cybersecurity functions and execute them autonomously and in parallel, enabling rapid host-level responses and coordinated multi-host defense.

The following sections describe the roles of each agent type and detail the structure and operational logic of the four swarms. To avoid repetition, the descriptions of the monitor and deployer agents are intentionally concise.

5.1. Monitor Agent

The monitor agent resides exclusively on the supervisor host and serves as the system's central strategic authority. Its primary responsibilities include:

- Maintaining a global situational model constructed from the reports received from all hosts.
- Orchestrating the deployment, scaling, and termination of swarms depending on the current defensive phase.
- Coordinating multi-host responses, including actions that require sequential or composite procedures.
- Ensuring consistency and enforcing trust boundaries by certifying that all deployment and reporting messages follow authenticated channels.
- Aggregating the semantic summaries sent by reporter agents and transforming them into a coherent, system-wide representation of the network state.
- Maintaining and periodically updating a local knowledge base of common service vulnerabilities, which is used to accelerate vulnerability identification by Network Defender swarm agents.

Operationally, the monitor implements both cyclic behaviors and one-shot behaviors. Cyclic behaviors evaluate incoming reports and detect state changes of interest. In contrast, one-shot behaviors are triggered by specific conditions, such as issuing a deployment request or initiating a forensic sweep. Algorithm 1 shows the monitor agent strategic control loop.

5.2. Deployer Agents

A deployer agent runs on each operational host and functions as the local executor of the monitor's strategic directives. Its responsibilities are intentionally limited in scope and centre primarily on swarm instantiation, that is, launching the specified swarm upon receiving an authenticated deployment request.

Deployer activity is event-driven: it reacts exclusively to deployment instructions from the monitor, with optional lightweight periodic checks restricted to verifying the status of previously launched swarms. Once a swarm has been instantiated, the deployer does not participate in its coordination or decision-making. Instead, the swarm agents interact directly with the monitor whenever supervision, reporting, or additional tasking is required.

Algorithm 1: Monitor Agent Strategic Control Loop

Input: Incoming semantic reports from Deployer agents
Data: Global situational model, Vulnerability Knowledge Base

```

1 Initialize global_state
2 Initialize knowledge_base
3 while system_is_active do
4   if operator_command_received() then
5     // Manual intervention override
6     deploy_swarm (operator_target)
7   end
8   messages ← receive_agent_reports()
9   update_situational_model (messages)
10  if detect_incident (global_state) then
11    // Autonomous response orchestration
12    deploy_specialized_swarm (threat_type, target_host)
13    execute_coordinated_response (global_state)
14  end
15  refine_knowledge_base (global_state)
16 end

```

5.3. Worker Agents

Worker agents are lightweight, specialised entities organised into four swarms, each focused on a distinct defensive function. Workers within a swarm exchange context-aware information, and they may also share data with workers of other swarms on the same host when it is beneficial. This behavior promotes emergent, bottom-up cooperation without centralising local decision-making.

The swarms are:

- Network Defender Swarm;
- Host Defender Swarm;
- Anomaly Detection Swarm;
- Forensic and Recovery Swarm.

While all swarms contribute essential capabilities, the Network Defender Swarm is presented in this paper as an illustrative example of the swarm design and operational logic. Its structure, interactions, and internal composition are described in detail below to demonstrate the principles applied across all swarms.

5.4. Network Defender Swarm

The Network Defender Swarm identifies the exposure surface of each operational host and detects network-level weaknesses that adversaries could exploit. Its internal organisation follows a pipeline-oriented model in which each agent type performs a specialised task and passes structured information to the next stage. This division of labour maximises parallelism and enables fine-grained control over how network reconnaissance is conducted within each host.

The swarm is instantiated on demand by the local deployer in response to a directive from the monitor. The swarm is composed of four types of worker agents: the scan agent, version miner agent, vulnerability checker agent, and network reporter agent. Their behaviors and interactions are described below.

The scan agent constitutes the first stage of the defensive pipeline and is responsible for enumerating the host's externally reachable services (see Algorithm 2). Its behavior

periodically executes a port-scanning routine, operating either in a focused mode (only standard ports) or in an exhaustive mode (all ports), as requested by the monitor. In addition to detecting open ports, the agent records the exposure level of each service, distinguishing whether it is bound to all interfaces (0.0.0.0:X), a specific external IP (X.Y.Z.W:P), or a loopback address (127.0.0.0/8). Ports not previously observed or bound to unexpected interfaces trigger notifications to the monitor. Once scanning is complete, the results are forwarded to the corresponding version miner agents through a direct host-local message exchange.

Algorithm 2: ScanAgent Reactive Behavior

Input: Scan parameters from Deployer agent
Output: Service state updates for local collaboration

```

1 Initialize local_service_state
2 while periodic_scan_active do
3   scan_scope ← receive_scan_parameters_from_deployer()
4   open_ports ← perform_port_scan(scan_scope)
5   foreach port in open_ports do
6     exposure ← determine_exposure(port)
7     update_local_service_state(port, exposure)
8   end
9   // Inter-agent coordination at host level
10  send_results_to_version_miner(local_service_state)
11 wait for next scan_cycle
12 end
  
```

Upon receiving a list of open ports from a scan agent, the version miner agent attempts to infer the software version associated with each detected service using passive banner-based analysis. Its behavior processes the service banners collected during the scanning phase and applies service-specific pattern matching to extract version information whenever possible. All extracted version strings are normalised into a consistent representation and forwarded to the vulnerability checker agents for subsequent assessment (see Algorithm 3).

Algorithm 3: VersionMinerAgent Reactive Behavior

Input: Service scan results from ScanAgent
Data: Local version heuristics, normalization rules
Output: Normalized version state for vulnerability analysis

```

1 Initialize local_version_state
2 while receiving_scan_results do
3   foreach service in scan_results do
4     banner ← extract_banner(service)
5     version ← infer_version(banner)
6     normalized_version ← normalize_version(version)
7     update_local_version_state(service, normalized_version)
8   end
9   // Forwarding intelligence to the next swarm agent
10  send_results_to_vulnerability_checker(local_version_state)
11 end
  
```

The vulnerability checker agent performs a host-local vulnerability assessment by matching the software versions collected by the version miner agent against a CVE-indexed knowledge base (CWE, NVD, exploitDB, etc.) made available at deployment time (see Algorithm 4). This knowledge base contains only the most commonly used services and version entries to minimize distribution and storage overhead. When the agent encounters a software version that is not present in the local subset, it issues an on-demand query to the NVD to retrieve the corresponding vulnerability metadata. Newly retrieved entries may then be cached locally or forwarded to the monitor for subsequent redistribution. The vulnerability checker performs semantic version-based lookups and generates a list of potential exposures, annotated with severity scores and exploitability metadata. The resulting vulnerability set is then forwarded to the reporter agent.

Algorithm 4: VulnerabilityCheckerAgent Reactive Behavior

Input: Normalized version data from VersionMinerAgent
Data: Local vulnerability cache, NVD API access
Output: Annotated vulnerability state for the Reporter agent

```

1 Initialize local_vuln_cache
2 while receiving version_data do
3   foreach (service, version) in version_data do
4     // Optimization: prioritize local knowledge
5     if version is in local_vuln_cache then
6       | vulnerabilities ← lookup_local (version)
7     else
8       | vulnerabilities ← query_NVD (version)
9       | update_local_vuln_cache (version, vulnerabilities)
10    end
11    annotated_vulns ← annotate_with_severity_and_exploitability
      (vulnerabilities)
12    update_local_vulnerability_state (service, annotated_vulns)
13  end
14 // Handoff to the host-level reporter
15 send_results_to_reporter (local_vulnerability_state)
16 end
  
```

The network reporter agent consolidates all findings from the Network Defender Swarm into a coherent, semantically normalised report for the monitor. Its behavior collects service enumerations, version mappings, and vulnerability assessments from the preceding agents and merges them into a single per-host network-exposure summary. The reporter validates message integrity, resolves conflicting information, and removes redundant entries arising from concurrent scans. Once assembled, the report is encoded using the system's standard ontology and transmitted to the monitor. The reporter forms the swarm's final output stage, ensuring that the monitor receives a clean, consistent representation of the host's network posture.

Together, these four agent types implement a fully autonomous network-defense pipeline capable of rapidly detecting newly exposed services, identifying outdated or vulnerable software, and forwarding actionable intelligence to the strategic layer.

5.5. Host Defender Swarm

The Host Defender Swarm preserves the internal integrity of each operational host. It inspects local execution artefacts, file-system state, and configuration parameters for

indicators of malicious activity or unsafe conditions. In contrast to the Network Defender Swarm, which targets externally exposed services, the Host Defender Swarm operates exclusively within the host boundary, focusing on threats arising from local execution. Its primary objective is to identify suspicious conditions early and report them before they escalate into full host compromise or lateral movement.

The swarm comprises multiple specialised worker agents, each addressing a distinct dimension of host-level security. The process and file auditor agent inspects running processes and selected areas of the file system to detect potentially malicious artefacts, such as unexpected binaries, anomalous process attributes, or files matching known malicious patterns. The configuration assessor agent evaluates operating system and application configuration states, identifying insecure settings, deviations from hardened baselines, or changes that may facilitate privilege escalation or persistence. A dedicated host reporter agent aggregates the findings produced by the inspection agents, resolves overlaps or inconsistencies, and generates a semantically normalised report for the monitor.

Through the cooperation of these agents, the Host Defender Swarm produces a multi-perspective assessment of the host's internal security posture without relying on intrusive techniques or actions that alter the operational environment. All inspection activities are designed to be observational and minimally disruptive, prioritising deployability in production-like settings.

The Host Defender Swarm is explicitly designed to be extensible. Additional inspection agents may be incorporated as new threat models or operational requirements emerge. This modular design ensures that the swarm can evolve while maintaining a unified communication and reporting interface toward the strategic layer.

5.6. Forensic and Remediation Swarm

The Forensic and Remediation Swarm constitutes the reactive operational layer of the architecture and is activated when other swarms report security-relevant findings. While the Network Defender, Host Defender, and Anomaly and Baseline swarms focus on detection and assessment, the Forensic and Remediation Swarm executes controlled response actions once sufficient evidence of risk, misconfiguration, or compromise has been established.

The swarm is designed to operate in close coordination with detection-oriented swarms and is only engaged upon explicit notification. Its responsibilities include isolating or quarantining artefacts identified as potentially malicious, preserving forensic evidence for subsequent analysis, and initiating carefully scoped remediation actions. These may involve applying security patches for well-known high-severity vulnerabilities, disabling vulnerable services, or enforcing hardened configuration states. All actions are designed to be minimally intrusive, auditable, and reversible, thereby preserving operational continuity while preventing the propagation or escalation of threats.

The swarm is composed of specialised worker agents, each responsible for a distinct response function. A quarantine agent isolates files or artefacts flagged by other swarms, preventing execution or further interaction while maintaining recoverability. A patch and upgrade agent is responsible for applying automated updates or security fixes when the impact and risk are well understood; in ambiguous or high-impact scenarios, it escalates the issue by issuing alerts rather than executing changes autonomously. A recovery assessor agent synthesises the available evidence and produces human-readable recommendations outlining suggested recovery or mitigation steps. As with other swarms, a dedicated forensic reporter agent consolidates all actions and outcomes into a semantically normalised report for the monitor.

By separating detection from response and enforcing explicit activation and verification steps, the Forensic and Remediation Swarm provides an autonomous yet controlled remediation capability. This design enhances system resilience against confirmed threats while maintaining transparency, auditability, and alignment with best practices in operational security.

5.7. Anomaly and Baseline Swarm

The Anomaly and Baseline Swarm operates as a continuous monitoring layer. Its primary objective is to characterise the standard behavioral patterns of each operational host and to detect deviations that may indicate abnormal or potentially malicious activity. Rather than focusing on explicit signatures or predefined rules, this swarm incrementally builds behavioral baselines derived from the system's routine operation. The monitored features may include resource usage trends, process lifecycles, network interaction patterns, and configuration stability over time.

The swarm is composed of specialised worker agents with clearly delineated roles. A system profiler agent collects low-level telemetry and periodically summarizes relevant behavioral indicators. An anomaly detector agent uses this information to construct and update lightweight baseline models, then compares incoming observations against them. When sustained or statistically significant deviations are observed, the agent generates anomaly alerts rather than definitive classifications. These alerts are then forwarded to an anomaly reporter agent, which consolidates and normalises them before dissemination.

Anomaly reports produced by this swarm are shared with the Network Defender, Host Defender, and Forensic and Remediation swarms, where they serve as contextual signals that may trigger deeper inspections or controlled response actions. The Anomaly and Baseline Swarm does not directly initiate remediation and does not assume that all deviations correspond to malicious behavior; instead, it provides early-warning indicators that complement the more deterministic analyses performed by other swarms.

Given the inherently broad scope of behavioral monitoring, the internal models and features employed by this swarm are intentionally designed to be lightweight and extensible. While future iterations may incorporate more advanced temporal modelling, adaptive thresholds, or cross-host correlation mechanisms, the present architecture positions the Anomaly and Baseline Swarm as a foundational component. It supports continuous, context-aware situational awareness without introducing excessive computational overhead or operational risk.

To facilitate an understanding of the agent organisation and their interactions, Table 3 summarizes the main swarms defined in the architecture, their primary responsibilities, the types of agents they comprise, and their principal communication relationships.

Table 3. Summary of agent swarms, roles, and agent types.

Swarm	Primary Role	Agent Types
Network Defender Swarm	Network exposure assessment and service-level vulnerability detection	scan agent, version miner agent, vulnerability checker agent, network reporter agent
Host Defender Swarm	Host-level integrity assessment and detection of malicious artefacts	process and file auditor agent, configuration assessor agent, host reporter agent
Anomaly and Baseline Swarm	Behavioral baseline construction and anomaly signalling	system profiler agent, anomaly detector agent, anomaly reporter agent
Forensic and Remediation Swarm	Controlled response, containment, and recovery actions	quarantine agent, patch and upgrade agent, recovery assessor agent, forensic reporter agent

6. Materials and Methods

This section describes the experimental environment used to evaluate the proposed Multi-Agent System and presents the implementation details of the Network Defender Swarm. This swarm was selected as a representative component since it performs measurable, time-critical defensive tasks related to attack surface characterisation and vulnerability assessment on operational hosts.

The evaluation was conducted in a controlled virtualised environment designed to emulate an operational host within a defended network. A virtual machine running Parrot OS was selected due to its stability, security-oriented software ecosystem, and compatibility with network inspection and analysis tools [48]. The SPADE multi-agent platform (version 4) was deployed on this machine, serving as both the agent execution framework and the communication middleware.

All agent interactions were handled through SPADE's built-in XMPP server (Pyjabber v0.3.0), which provided a fully isolated messaging layer bound exclusively to the local loopback interface (127.0.0.1). The experimental setup was organised into two concurrent terminals: one dedicated exclusively to running SPADE and the XMPP server, and another used to launch the main script that initialises the Multi-Agent System and deploys the Network Defender Swarm. This separation ensured reproducible communication conditions and a clear distinction between middleware execution and agent logic.

The Network Defender Swarm was implemented as a host-local processing pipeline. It efficiently characterises the attack surface through parallel, specialised tasks. Internally, the swarm has three consecutive functional stages: service discovery, version extraction, and vulnerability verification.

The vulnerability checker agent relies on the Common Vulnerabilities and Exposures (CVE) system as its primary knowledge base. CVE was selected due to its status as a globally recognized standard for vulnerability identification and tracking [49].

Algorithm 5 provides a high-level overview of the Network Defender Swarm workflow. The process is organised into four main stages: service discovery (ScanAgent), intelligence enrichment (VersionMinerAgent), risk assessment (VulnerabilityCheckerAgent), and handoff to the monitor (NetworkReporterAgent).

The widely used network scanner nmap (version 7.93) was employed as an orientative baseline to contextualize the Total Detection Time measured for the Network Defender Swarm. The selected configuration, `nmap -sV -p- 127.0.0.1`, performs a full TCP port scan over the entire port range (1–65,535) while retrieving service version information. Each nmap experiment was repeated multiple times to account for execution variability, and the reported Total Detection Time (TDT) corresponds to the arithmetic mean and standard deviation across all runs.

All experiments recorded several key metrics to evaluate system performance. Total Detection Time (TDT) measured the elapsed time from the deployment command until the consolidated report was received. ACL message latency was measured by recording the total time required for a predefined set of messages to be sent and received between agents, including serialization, XMPP transport, and inbox processing. Coordination overhead was assessed by quantifying the additional time incurred as the number of agents increased, reflecting the cost of task allocation and result aggregation. Each experimental configuration was repeated at least 5 times, and the resulting means and standard deviations were computed to ensure statistical reliability.

Two main experimental scenarios were explored. In the first one, the scalability of the ScanAgent was evaluated by partitioning the TCP port space across varying numbers of agents, ranging from 2 to 256. It allows the assessment of TDT and coordination overhead under increasing parallel workloads. In the second scenario, messaging efficiency

was examined by increasing the number of agents engaged in ACL message exchanges, with each agent sending 5000 messages. These results were compared against JADE 4.6 measurements reported in the literature [30] to provide context on communication latency and scalability trends.

Algorithm 5: NetworkDefenderSwarm Orchestration

Input: Target host network interfaces
Data: Local DB, NVD Access, Connection limits (e.g., 300)
Output: Comprehensive host vulnerability report sent to Monitor

```

1 Initialize host_state
  // Stage 1: Service Discovery (ScanAgent)
2 open_ports ← scan all TCP ports concurrently
3 foreach successful connection do
4   | banner ← grab_service_banner()
5   | service_type ← infer_service_type(port, banner)
6 end
  // Stage 2: Intelligence Enrichment (VersionMinerAgent)
7 foreach service in open_ports do
8   | version ← extract_version_from_banner(service)
9   | cache_version(service, version)
10 end
  // Stage 3: Risk Assessment (VulnerabilityCheckerAgent)
11 foreach (service, version) in version_data do
12   | if version is in local_db then
13   |   | vuln_info ← lookup_local(version)
14   | else
15   |   | vuln_info ← query_NVD(version)
16   |   | update_local_db(version, vuln_info)
17   | end
18 end
  // Stage 4: Handoff to the Monitor (NetworkReporterAgent)
19 send_results_to_monitor(vulnerabilities)
  
```

To support reproducibility, the complete source code used in this study will be uploaded to a public repository (<https://github.com/lusilux/cyberSPADE.git>, accessed on Feb. 5th 2026) prior to publication, including the implementations of all agents, execution scripts, and auxiliary files required to reproduce the experiments.

7. Results

The following section presents an initial, exploratory evaluation of the Network Defender Swarm component of cyberSPADE. The primary objective is to illustrate how the system supports distributed, cooperative multi-agent coordination in host-level network tasks. Experiments are intended to validate the architecture and messaging mechanisms rather than to demonstrate superior detection performance.

All experiments were conducted in a controlled laboratory environment characterised by localhost communication and low agent density. The reported results are therefore not intended to demonstrate large-scale performance gains, but rather to illustrate the potential benefits of parallelism, coordination, and asynchronous execution inherent to communicative multi-agent architectures. A comprehensive validation across multiple hosts or distributed virtual machines is left for future work.

7.1. Scan Agent Scalability Analysis

To evaluate coordination under increasing parallel workloads, a set of experiments was conducted in which the scanning task was partitioned among multiple scan agents. This setup illustrates how cyberSPADE manages task allocation and coordination overhead.

In this configuration, the total TCP port range (1–65,535) was evenly divided among the deployed scan agents. The selection of these specific thresholds (2, 4, 6, 8, 16, 32, 64, 128, 256 agents) enables direct comparison with established multi-agent platform benchmarks, particularly the JADE performance evaluation framework [30]. Each agent independently scanned its assigned port subset using the same asynchronous, non-blocking connection strategy described previously. All agents operated concurrently and forwarded their partial results to a single receiver agent, which aggregated the data and triggered the remaining stages of the analysis pipeline.

The results presented in Table 4 and in Figure 4 reveal a linear relationship between the number of scan agents and the observed Total Detection Time. While distributing the scanning workload among multiple agents significantly reduces detection time, increasing the number of scan agents beyond a specific threshold does not yield proportional improvements and, eventually, leads to performance degradation. This behavior can be attributed to coordination overhead, including increased message traffic to the receiver agent and contention for shared system resources, such as CPU scheduling and socket management.

Table 4. Impact of scan agent parallelism on Total Detection Time (TDT) in seconds.

Number of Scan Agents	Total Detection Time (TDT)
1	11.89
2	15.12
4	13.25
6	13.07
8	14.05
16	12.88
32	23.00
64	36.88
128	69.72
256	121.37

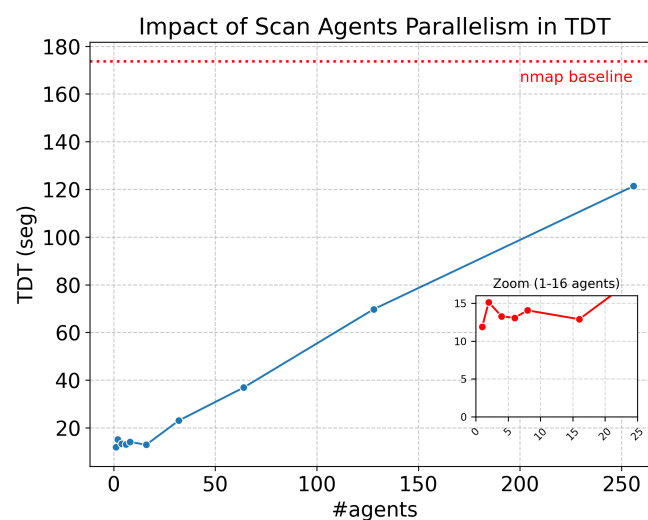


Figure 4. Impact of Scan Agent Parallelism on Total Detection Time (TDT). The horizontal dashed line at 173.66 s represents the average TDT of the traditional monolithic baseline (nmap), as reported in Table 5, and is included for contextual reference.

As shown in the data, the system achieves peak efficiency between 2 and 16 agents, with the lowest detection time of 12.88 s recorded for the 16-agent configuration. However, a turning point occurs after 16 agents. From this point forward, the Total Detection Time increases significantly, reaching 36.88 s with 64 agents and peaking at 121.37 s with 256 agents. This upward trend can be attributed to coordination overheads and shows the physical limit of the host platform. When a high number of agents are deployed on a single machine, the system encounters a bottleneck in shared resource management.

These findings suggest that the Network Defender Swarm benefits most from moderate levels of parallelism (up to 16 agents), where workload partitioning and asynchronous execution significantly reduce detection latency without incurring excessive coordination overhead.

Table 5. Baseline network scan results using nmap with the configuration -sV -p- 127.0.0.1, performing a full TCP port scan with service version detection..

Metric	Value
Scan configuration	nmap -sV -p- 127.0.0.1
Total execution time (TDT) in seconds	173.66 ± 5.32
TCP ports scanned	65,535
Services identified	6
Scanning paradigm	Monolithic, sequential

7.2. Messaging Efficiency: SPADE vs. JADE

To assess the point-to-point messaging efficiency of our SPADE-based Multi-Agent System, we conducted a benchmark measuring the average time for ACL message exchanges between agents on the same host. This metric captures the time required for a set of agents to serialize a message, deliver it to the underlying XMPP runtime, and receive it at the recipient's inbox, including user-level processing.

This methodological choice enables a direct comparison with the classic performance study “Efficiency of the JADE Agent Platform” [30]. Although the original work evaluates large distributed deployments involving multiple machines, heterogeneous hardware, and hundreds of thousands of ACL messages, it also reports the per-message average sending and receiving times, obtained by dividing the total spamming/processing time by the number of messages (5000 per agent). These normalized values constitute the closest available equivalent metric for comparison with a local SPADE benchmark.

It is essential to note that the remaining results of the JADE study (e.g., multi-machine throughput, container-level saturation, and processing bottlenecks under heterogeneous hardware) cannot be directly compared with a local benchmark, as they arise from distributed system effects not present in our experimental setup. For this reason, the comparison is strictly limited to per-message ACL latency.

In our experiment, the scan agents sent 5000 ACL messages of 300 bytes each to the miner agent, all running on the same host. The benchmark measured (i) total spamming time (serialization + delivery through the XMPP transport) and (ii) total receiving time (retrieval from the agent's inbox + user-level processing). These aggregated measurements are directly comparable to the values reported in the JADE study for the same workload size.

The results are summarized in Table 6, which reports total spamming and receiving times for increasing numbers of interacting agents. Although absolute timing values vary significantly due to differences in hardware, middleware architecture, and execution environment, the table enables a direct comparison of scalability trends and message processing overhead as communication load increases.

When comparing these normalized values, cyberSPADE demonstrates reductions of approximately one to two orders of magnitude relative to the latencies reported for JADE.

These results indicate a substantially lower message-processing overhead in SPADE within the evaluated experimental conditions. This behavior can be attributed to its asynchronous execution model and integration with the XMPP runtime. While this comparison does not aim to provide a definitive performance ranking across agent platforms, it supports SPADE's suitability for communication-intensive multi-agent architectures where low per-message latency is desirable.

Table 6. ACL Message Latency Comparison: JADE vs. cyberSPADE (Total Time for 5000 Messages, time in ms.).

Agents	JADE Spamming	JADE Receiving	cyberSPADE Spamming	cyberSPADE Receiving
2	40,034	87,053	86	502
3	24,440	141,778	90	525
4	25,128	217,501	87	603
5	25,217	313,625	81	661
6	28,843	448,181	98	798
7	35,164	634,847	89	897
8	40,624	821,341	96	10,278

7.3. Comparison with MAS and Distributed Agent Benchmarks from the Literature

This subsection contextualizes the ACL messaging latency and coordination overhead observed in cyberSPADE by comparing them with quantitative benchmarks reported for representative multi-agent and distributed agent frameworks in the literature. The scope of this comparison is explicitly restricted to communication and coordination-level behavior, and does not include application-level performance, task execution time, or end-to-end workload latency.

In addition to JADE, two further frameworks are considered in the following benchmark-oriented comparison: CloneMAP and DynTaskMAS. They represent distributed agent and task-execution frameworks that report quantitative metrics directly relevant to communication latency, coordination overhead, and scalability as agent populations increase. CloneMAP [50] is a large-scale distributed agent replication framework designed to support massive agent populations through dynamic cloning and state replication mechanisms. DynTaskMAS [51] is a dynamic task-oriented multi-agent framework focused on adaptive scheduling and load balancing under increasing agent loads.

Although no prior work reports ACL message benchmarks under identical middleware, workload, and hardware conditions, several studies provide closely related metrics capturing message exchange latency under increasing agent populations. These metrics represent the closest available references for assessing whether the communication and coordination behavior exhibited by cyberSPADE is consistent with that of established MAS and distributed agent platforms.

Table 7 summarizes indicative results from four platforms: JADE, cyberSPADE, CloneMAP, and DynTaskMAS. The reported metrics are not strictly equivalent, as they reflect different abstraction levels, workloads, and experimental environments.

Table 7. Indicative comparison of communication and coordination latency metrics across MAS platforms. JADE and cyberSPADE values correspond to aggregate ACL send and receive times reported in Table 6. CloneMAP values correspond to the average round-trip time (avgRTT) reported in [50]. DynTaskMAS values correspond to end-to-end task completion times reported in [51].

Platform	Metric Type	Agents	Value (ms)
JADE	ACL send + receive latency	2–8	40,000–821,000
cyberSPADE	ACL send + receive latency	2–8	90–1030
CloneMAP	avgRTT	10,000	11
DynTaskMAS	End-to-end task time	4–16	3700–4700

The most direct comparison is between JADE and cyberSPADE, as both report ACL-level send and receive latencies under comparable workload conditions (5000 messages per agent across 2–8 agents). CyberSPADE demonstrates latencies that are approximately one to two orders of magnitude lower than those reported for JADE. This can be attributed to cyberSPADE’s asynchronous execution model and the use of lightweight XMPP-based messaging, as discussed in the previous subsection.

CloneMAP’s reported average round-trip time of 11 milliseconds for 10,000 agents represents a fundamentally different operational regime. This low latency reflects CloneMAP’s optimization for massive-scale deployments where middleware-level message passing must support high-frequency state synchronization across thousands of agent clones. This metric captures the messaging infrastructure under minimal agent behavior, using a lightweight execution and messaging model. In contrast, cyberSPADE’s reported latencies include the full cost of ACL message construction, XMPP stanza serialization, and user-level inbox processing.

DynTaskMAS reports end-to-end task completion times ranging from 3.7 to 4.7 s for 4–16 agents. These values represent the total time required to decompose, allocate, execute, and aggregate results for application-level tasks, and therefore include both coordination overhead and computational work. A direct numerical comparison between these metrics and cyberSPADE’s messaging latencies is not meaningful, as the metrics capture different abstraction levels. Nevertheless, the comparison remains informative in that it shows cyberSPADE’s coordination overhead (on the order of 100–1000 milliseconds per message exchange) constitutes only a small fraction of the end-to-end task execution times typically reported in task-oriented MAS.

Taken together, these comparisons indicate that the messaging latency and coordination behavior observed in cyberSPADE fall within the expected performance envelope of modern asynchronous MAS and distributed agent platforms operating under tightly coupled or host-local conditions.

The platform demonstrates three key characteristics that position it within the modern MAS landscape. First, cyberSPADE achieves substantial improvements over classical synchronous platforms, with lower latencies than JADE, validating the benefits of asynchronous execution and native XMPP integration. Second, while CloneMAP achieves lower absolute latencies through minimalist messaging, cyberSPADE operates at a higher semantic level, enabling richer coordination protocols necessary for autonomous cyberdefense. Third, the messaging latencies remain small relative to typical task execution times, ensuring that coordination overhead does not dominate performance in realistic defensive scenarios.

The observed messaging latencies reflect a deliberate design choice that prioritizes semantically rich, asynchronous coordination over minimal middleware overhead, while remaining well within the performance bounds required by realistic cyberdefense workloads. Overall, these findings support the suitability of cyberSPADE as an experimental testbed

for evaluating coordination mechanisms and communication protocols in autonomous cyberdefense scenarios. At the same time, they point to clear directions for future work, particularly the evaluation of scalability and performance under distributed, multi-host deployments with heterogeneous network conditions and larger agent populations.

7.4. Baseline Detection Experiment Using a Traditional Scanner

This subsection provides a contextual baseline intended to illustrate the operation differences between a conventional monolithic network scanner and the Network Defender Swarm architecture. The experiment is not intended as a competitive performance evaluation, but rather as a reference point to help interpret the end-to-end detection times observed in the proposed system.

To establish a reference baseline, the widely adopted network scanning tool *nmap* was selected due to its extensive use in operational security assessments and research. The selected configuration (*nmap -sV -p- 127.0.0.1*) performs a full TCP port scan over the complete port range (1–65,535) combined with service version detection.

When service detection (*-sV*) is enabled, *nmap* typically establishes full TCP connections to open ports in order to perform application-layer fingerprinting. This process involves a standard TCP three-way handshake followed by protocol-specific probes, adaptive retransmissions, and timeout management. While this approach yields highly reliable service identification, it can introduce noticeable latency during exhaustive scans, particularly across diverse network conditions.

To account for execution variability, the *nmap* experiment was repeated multiple times under identical conditions. The reported Total Detection Time (TDT) corresponds to the arithmetic mean across all runs.

As shown in Table 5, the average execution time of 173.66 s reflects the trade-off inherent to *nmap*'s design: high-fidelity service identification at the cost of extended execution time when probing the whole TCP port space.

The equivalent experiment was then conducted using the Network Defender Swarm (see Table 8). In this configuration, the monitor instructs the deployer to instantiate the swarm, after which the scan agent performs host-local service discovery and forwards its findings to the version miner agent. Upon reception of all partial scan results, the miner aggregates the data, derives service version information, and forwards the consolidated output to the remaining stages of the security analysis pipeline.

Unlike *nmap*'s centrally orchestrated workflow, the scan agent is implemented using Python's *asyncio* infrastructure and executes non-blocking TCP connection attempts. A semaphore limits concurrency to a maximum of 300 simultaneous connection attempts. This threshold was empirically determined to balance scanning performance with system resource constraints. Higher concurrency levels risk socket exhaustion and operating system scheduler saturation, while lower values unnecessarily extend scanning duration.

Each connection attempt relies on standard TCP connection establishment and therefore entails a full three-way handshake. However, unlike *nmap*, the scan agent deliberately avoids extensive protocol fingerprinting, adaptive timing logic, and retransmission strategies. Instead, it employs lightweight banner-grabbing heuristics tailored to standard services, prioritizing speed over exhaustive protocol analysis.

For the MAS-based approach, the Total Detection Time (TDT) is conservatively defined as the elapsed time from the moment the monitor agent issues the deployment command until the consolidated report is received from the reporter agent. As with the baseline, measurements were averaged across multiple runs.

The observed performance difference is summarised in Table 9.

The difference in observed detection time primarily reflects architectural and methodological differences between the two approaches. While the Network Defender Swarm completes host-level reconnaissance significantly faster under the evaluated conditions, this result should be interpreted in light of the differing objectives of each system.

Specifically, the multi-agent approach benefits from parallel execution and the decomposition of the scanning workflow into cooperative stages. In contrast, nmap emphasizes completeness, protocol correctness, and robustness across diverse network environments, which naturally increases execution time during exhaustive scans.

It is worth noting that the observed performance differences are specific to the evaluated configuration and workload, and do not account for factors such as network latency, packet loss, or heterogeneous host capabilities, which may influence behavior in real-world deployments.

Table 8. Network Defender Swarm scan results. The configuration uses four specialized agents: Scan Agent, Version Miner Agent, Vulnerability Checker Agent, and Reporter Agent.

Metric	Value
Scanning configuration	Network Defender Swarm
Total Detection Time (TDT) in seconds	11.89 ± 0.33
TCP ports scanned	65,535
Services identified	6
Execution paradigm	Distributed, cooperative, parallel

Table 9. Total Detection Time comparison for full port coverage (65,535) on a local host.

Evaluated System	Total Detection Time (TDT) in Seconds
Nmap (Monolithic baseline)	173.66 ± 5.32
Network Defender Swarm (MAS)	11.89 ± 0.33

8. Discussion

The experimental results presented in this work provide an initial indication of the potential advantages of a communicative Multi-Agent System for autonomous cyberdefense, while also revealing several limitations inherent to the experimental setup. The evaluation was conducted in a controlled, small-scale environment in which all agents operated on a single host using SPADE's local XMPP runtime. This configuration enabled the isolation of the Network Defender Swarm's behavior and facilitated the assessment of baseline performance characteristics—such as coordination dynamics, task partitioning and intra-agent message latency—without the confounding effects of network variability or distributed deployment.

The comparison between the MAS-based detection pipeline and the traditional monolithic nmap scanner serves primarily as a contextual reference. While the Network Defender Swarm completed host-level reconnaissance significantly faster under the evaluated conditions, this difference reflects fundamental architectural and methodological distinctions rather than a direct measure of detection effectiveness.

Similarly, the message exchange benchmark revealed that SPADE achieves considerably lower per-message processing latency than the JADE baselines reported in the literature. These results reinforce SPADE's suitability for communication-intensive MAS designs. The absence of network heterogeneity, routing overhead, XMPP server load, and competing message queues means that the measured latencies represent an optimistic lower bound rather than a generalisable estimate of performance in large or distributed MAS deployments.

A further limitation concerns the simplification of specific components of the architecture. Although the Network Defender Swarm was implemented as a whole, other swarms within the proposed system exist only at a conceptual level. Likewise, the secure monitor–deployer communication protocol, although fully designed, was not yet operationalized in the experimental prototype. As a result, the current evaluation cannot assess end-to-end performance, security guarantees, or fault-tolerant behavior across the entire architecture.

Future work will therefore focus on three main directions. First, the full implementation of all agent types and behaviors is required to enable an end-to-end assessment of the architecture, particularly regarding coordination dynamics among heterogeneous swarms and the integration of real-time anomaly detection and remediation mechanisms. In second place, the secure communication system must be completed and experimentally validated, including TOTP-based onboarding, symmetric session establishment, and optional post-quantum key exchange. This scheme will allow empirical measurement of cryptographic overhead and resilience under adversarial interference. Finally, the experimental environment will be expanded to multi-host, heterogeneous, and high-density deployments to evaluate performance in realistic distributed conditions. Such scenarios will enable the study of XMPP routing behavior, message queue contention, scalability limits, and the impact of differing hardware capabilities on swarm responsiveness.

In summary, the present results provide an initial architectural validation of cyberSPADE and demonstrate the feasibility of using a cooperative, communication MAS for host-level cyberdefense tasks. While the current results are intentionally limited in scope, they establish a foundation for future large-scale experimentation to assess the robustness, scalability, and operational viability of the proposed architecture in realistic cyber-range environments.

Author Contributions: Conceptualization, L.A.T., M.R., J.P. and M.A.L.; methodology, L.A.T.; software, L.A.T.; validation, L.A.T.; formal analysis, L.A.T.; investigation, L.A.T., M.R., J.P. and M.A.L.; writing—original draft preparation, L.A.T.; writing—review and editing, M.R. and J.P.; supervision, M.R., J.P. and M.A.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by project PID2024-158227NB-C33 funded by MICIU/AEI/10.13039/501100011033/ FEDER, UE, and by the Valencian Government through grant CIPROM/2021/077.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Finistrella, S.; Mariani, S.; Zambonelli, F. Multi-Agent Reinforcement Learning for Cybersecurity: Classification and Survey. *Intell. Syst. Appl.* **2025**, *26*, 200495.
2. Okoli, U.I.; Obi, O.C.; Adewusi, A.O.; Abrahams, T.O. Machine Learning in Cybersecurity: A Review of Threat Detection and Defense Mechanisms. *World J. Adv. Res. Rev.* **2024**, *21*, 2286–2295. <https://doi.org/10.30574/wjarr.2024.21.1.0315>.
3. Komaragiri, V.B.; Edward, A. AI-Driven Vulnerability Management and Automated Threat Mitigation. *Int. J. Sci. Res. Manag.* **2022**, *10*, 980–998.
4. Landolt, C.R.; Würsch, C.; Meier, R.; Mermoud, A.; Jang-Jaccard, J. Multi-Agent Reinforcement Learning in Cybersecurity: From Fundamentals to Applications. *arXiv* **2025**, arXiv:2505.19837.
5. Saeed, I.A.; Selamat, A.; Rohani, M.F.; Krejcar, O.; Chaudhry, J.A. A Systematic State-of-the-Art Analysis of Multi-Agent Intrusion Detection. *IEEE Access* **2020**, *8*, 180184–180209.

6. Tyagi, A.K. Cyber Physical Systems (CPSs)—Opportunities and Challenges for Improving Cyber Security. *Int. J. Comput. Appl.* **2016**, *137*, 19–27.
7. Motwani, S.R.; Baranchuk, M.; Strohmeier, M.; Bolina, V.; Torr, P.H.S.; Hammond, L.; de Witt, C.S. Secret Collusion among AI Agents: Multi-Agent Deception via Steganography. *arXiv* **2025**, arXiv:2402.07510.
8. Peigné, P.; Kniejski, M.; Sondej, F.; David, M.; Hoelscher-Obermaier, J.; de Witt, C.S.; Kran, E. Multi-Agent Security Tax: Trading Off Security and Collaboration Capabilities in Multi-Agent Systems. *Proc. AAAI Conf. Artif. Intell.* **2025**, *39*, 27573–27581.
9. Owoputi, R.; Ray, S. Security of Multi-Agent Cyber-Physical Systems: A Survey. *IEEE Access* **2022**, *10*, 121465–121479.
10. Castro, S.R.; Campbell, R.; Lau, N.; Villalobos, O.; Duan, J.; Cardenas, A.A. Large Language Models are Autonomous Cyber Defenders. In Proceedings of the 2025 IEEE Conference on Artificial Intelligence (CAI), Santa Clara, CA, USA, 5–7 May 2025; IEEE: Piscataway, NJ, USA, 2025; pp 1125–1132.
11. Gomez, J.; Kfoury, E.F.; Crichigno, J.; Srivastava, G. A survey on network simulators, emulators, and testbeds used for research and education. *Comput. Netw.* **2023**, *237*, 110054.
12. Grimaldi, A.; Ribiollet, J.; Nespoli, P.; Garcia-Alfaro, J. Toward Next-Generation Cyber Range: A Comparative Study of Training Platforms. In *Computer Security, Proceedings of the ESORICS 2023 International Workshops, The Hague, The Netherlands, 25–29 September 2023*; Lecture Notes in Computer Science; Katsikas, S., Abie, H., Ranise, S., Verderame, L., Cambiaso, E., Ugarelli, R., Praça, I., Li, W., Meng, W., Furnell, S., et al., Eds.; Springer: Cham, Switzerland, 2024; Volume 14399.
13. Li, J.; Guo, X.; Zhao, Y.; Zhang, L.; Zhai, L. Poster: SpiderSim: Multi-Agent Driven Theoretical Cybersecurity Simulation for Industrial Digitalization. *arXiv* **2025**, arXiv 2502.13778.
14. Du, C.; Wang, C.; Chao, Y.; Xie, X.; Cui, Y. AI Agent Communication from Internet Architecture Perspective: Challenges and Opportunities. *arXiv* **2025**, arXiv 2509.02317.
15. Marro, S.; La Malfa, E.; Wright, J.; Li, G.; Shadbolt, N.; Wooldridge, M.; Torr, P. A Scalable Communication Protocol for Networks of Large Language Models. *arXiv* **2024**, arXiv 2410.11905.
16. Li, L.; El Rami, J.S.; Taylor, A.; Rao, J.H.; Kunz, T. Unified Emulation-Simulation Training Environment for Autonomous Cyber Agents. In *Machine Learning for Networking, Proceedings of the MLN 2022, Paris, France, 28–30 November 2022*; Lecture Notes in Computer Science; Renault, É., Mühlethaler, P., Eds.; Springer: Cham, Switzerland, 2023; Volume 13767.
17. Li, L.; El Rami, J.S.; Taylor, A.; Rao, J.H.; Kunz, T. Enabling A Network AI Gym for Autonomous Cyber Agents. In Proceedings of the 2022 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 14–16 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp 172–177.
18. Palanca, J.; Terrasa, A.; Julian, V.; Carrascosa, C. SPADE 3: Supporting the New Generation of Multi-Agent Systems. *IEEE Access* **2020**, *8*, 182537–182549.
19. Palanca, J.; Rincon, J.A.; Carrascosa, C.; Julian, V.; Terrasa, A. Flexible Agent Architecture: Mixing Reactive and Deliberative Behaviors in SPADE. *Electronics* **2023**, *12*, 659.
20. Garcia-Bohigues, M.; Cordova, C.; Taverner, J.; Palanca, J.; del Val, E.; Argente, E. Towards a Distributed Platform for Normative Reasoning and Value Alignment in Multi-Agent Systems. *arXiv* **2024**, arXiv 2405.13543.
21. Radhakrishnan, G.; Chithambaram, V.; Shunmuganathan, K. L. Comparative study of JADE and spade multi agent system. *Int. J. Adv. Res.* **2018**, *6*, 1035–1042.
22. Sunkara, G. Multi-Agent AI Systems for Coordinated Cybersecurity in Smart Cities. *Int. J. Res. Publ. Seminars* **2025**, *16*, 4–22.
23. Lage, O.; de Diego, S.; Urkizu, B.; Gómez, E.; Gutiérrez, I. Blockchain Applications in Cybersecurity. In *Computer Security Threats*; IntechOpen: London, UK, 2020.
24. Kiely, M.; Bowman, D.; Standen, M.; Moir, C. On Autonomous Agents in a Cyber Defence Environment. *arXiv* **2023**, arXiv:2309.07388.
25. Russell, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*, 4th ed.; Pearson: London, UK, 2021.
26. Wooldridge, M. *An Introduction to MultiAgent Systems*, 2nd ed.; John Wiley & Sons: Chichester, UK, 2009.
27. Stone, P.; Veloso, M. Multiagent Systems: A Survey from a Machine Learning Perspective. *Auton. Robots* **2000**, *8*, 345–383. <https://doi.org/10.1023/A:1008942012299>.
28. Standen, M.; Lucas, M.; Bowman, D.; Richer, T.; Kim, J.; Marriott, D. CybORG: A Gym for the Development of Autonomous Cyber Agents. *arXiv* **2021**, arXiv:2108.09118. <https://doi.org/10.48550/arXiv.2108.09118>.
29. Bellifemine, F.L.; Caire, G.; Greenwood, D. *Developing Multi-Agent Systems with JADE*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
30. Chmiel, K.; Gawinecki, M.; Kaczmarek, P.; Szymczak, M.; Paprzycki, M. Efficiency of JADE Agent Platform. *Sci. Program.* **2005**, *13*, 159–172.
31. Poslad, S.; Buckle, P.; Hadingham, R. The FIPA-OS Agent Platform: Open Source for Open Standards. In Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-2000), Manchester, UK, 10–13 April 2000; pp. 355–368.

32. Nwana, H.S.; Ndumu, D.T.; Lee, L.C.; Collis, J.C. ZEUS: A Toolkit for Building Distributed Multi-Agent Systems. *Appl. Artif. Intell.* **1999**, *13*, 129–185.
33. Palanca, J.; Rincon, J.; Carrascosa, C.; Julian, V.; Terrasa, A. A Flexible Agent Architecture in SPADE. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection, Proceedings of the PAAMS 2022, L'Aquila, Italy, 13–15 July 2022*; Lecture Notes in Computer Science; Dignum, F., Mathieu, P., Corchado, J.M., De La Prieta, F., Eds.; Springer: Cham, Switzerland, 2022; Volume 13616.
34. Bordini, R.H.; Hübner, J.F.; Wooldridge, M. *Programming Multi-Agent Systems in AgentSpeak Using Jason*; John Wiley & Sons: Hoboken, NJ, USA, 2007.
35. Wilensky, U. *NetLogo. Center for Connected Learning and Computer-Based Modeling*; Northwestern University: Evanston, IL, USA, 1999. Available online: <http://ccl.northwestern.edu/netlogo/> (accessed on Feb. 5th 2026).
36. Pokahr, A.; Braubach, L.; Lamersdorf, W. Jadex: A BDI Reasoning Engine. In *Multi-Agent Programming: Languages, Platforms and Applications*; Springer: Boston, MA, USA, 2005; pp. 149–174.
37. Fernandes, E.L.; Antichi, G.; Böttger, T.; Castro, I.; Uhlig, S. Faster Control Plane Experimentation with Horse. In *SIGCOMM Posters and Demos '19, Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos, Beijing, China, 19–23 August 2019*; ACM: New York, NY, USA, 2019; pp. 51–53.
38. Alliche, R.A.; da Silva Barros, T.; Aparicio-Pardo, R.; Sassatelli, L. Impact Evaluation of Control Signalling onto Distributed Learning-based Packet Routing. In *Proceedings of the ITC-34-Teletraffic Engineering for Smart Networking Conference, Shenzhen, China, 14–16 September 2022*.
39. Rezazadeh, F.; Gargari, A.A.; Lagén, S.; Song, H.; Niyato, D.; Liu, L. Toward Generative 6G Simulation: An Experimental Multi-Agent LLM and ns-3 Integration. In *Proceedings of the 2025 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Nice, France, 7–10 July 2025*; IEEE: Piscataway, NJ, USA, 2025; pp. 1–6.
40. Saint-Andre, P. *XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies*; O'Reilly Media: Sebastopol, CA, USA, 2009.
41. Malik, M.I.; McAteer, I. N.; Hannay, P.; Firdous, S. N.; Baig, Z. XMPP architecture and security challenges in an IoT ecosystem. In *Proceedings of the 16th Australian Information Security Management Conference, Perth, Australia, 5–6 December 2017*; Johnstone, M., Ed.; Security Research Institute, Edith Cowan University: Perth, Australia, 2018; pp. 62–73.
42. Weiss, G. (Ed.) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2013.
43. Parunak, H.V.D. *Practical and Industrial Applications of Agent-Based Systems*; Environmental Research Institute of Michigan (ERIM): Ann Arbor, MI, USA, 1998.
44. Singh, A.; Rathbun, E.; Graham, E.; Oakley, L.; Boboila, S.; Oprea, A.; Chin, P. Hierarchical Multi-agent Reinforcement Learning for Cyber Network Defense. *arXiv* **2024**, arXiv:2410.17351.
45. M'Raihi, D.; Machani, S.; Pei, M.; Rydell, J. *TOTP: Time-Based One-Time Password Algorithm*; RFC 6238; IETF: Fremont, CA, USA, 2011.
46. *FIPS 203; Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)*. NIST: Gaithersburg, MD, USA, 2024.
47. *FIPS 204; Module-Lattice-Based Digital Signature Standard (ML-DSA)*. NIST: Gaithersburg, MD, USA, 2024.
48. Parrot Project. *Parrot Security OS Documentation*; The Parrot Project: Southern, CA, USA, 2025. Available online: <https://parrotsec.org/docs/> (accessed on 16 Diciembre 2025).
49. Barchuk, B.; Volkov, K. Limitations of modern vulnerability scanners and CVE Systems. *World J. Adv. Eng. Technol. Sci.* **2024**, *12*, 973–989. <https://doi.org/10.30574/wjaets.2024.12.2.0348>.
50. Dähling, S.; Razik, L.; Monti, A. Enabling scalable and fault-tolerant multi-agent systems by utilizing cloud-native computing. *Auton. Agents-Multi-Agent Syst.* **2021**, *35*, 10. <https://doi.org/10.1007/s10458-020-09489-0>.
51. Yu, J.; Ding, Y.; Sato, H. DynTaskMAS: A Dynamic Task Graph-driven Framework for Asynchronous and Parallel LLM-based Multi-Agent Systems. In *ICAPS '25, Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025), Melbourne, Victoria, Australia, 9–14 November 2025*; Volume 35, pp. 288–296. <https://doi.org/10.1609/icaps.v35i1.36130>.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.