AI and Data Science Journal
Online ISSN: 3078-7394
DOI: https://doi.org/10.61784/adsj3034

AI-ASSISTED DISTRIBUTED RISK ASSESSMENT FOR ZERO-TRUST MICROSERVICES: A TAXONOMY AND CONCEPTUAL FRAMEWORK

DaoQuan Zhou[1*], XiongSheng Yi[2]
[1]*ITPM (IT Program Management), New England College, Henniker 03242, NH, USA.*
[2]*Department of Computer Science and Engineering, Santa Clara University, Santa Clara 95053, CA, USA.*
[*]*Corresponding Author: DaoQuan Zhou*

Abstract: Microservice and cloud-native architectures have enlarged the software attack surface faster than the methods used to assess its risk have adapted. Conventional vulnerability management remains centralised, static, focused on intrinsic severity, and disconnected from access control, none of which suits a distributed system in which risk depends on context and connectivity rather than on isolated flaws. This paper proposes a conceptual framework for distributed, AI-assisted risk assessment that addresses this mismatch. The framework is a synthesis of four research literatures, namely zero-trust architecture, microservice and cloud-native security, vulnerability analytics, and software risk management, and the paper first shows precisely where these literatures fail to meet. A six-dimensional taxonomy then organises the design space of distributed risk assessment, classifying approaches by signal source, granularity, analytical technique, risk model, zero-trust integration point, and temporality, and exposing the combination that no current approach occupies. To fill that gap, the paper proposes a four-layer framework in which lightweight per-service agents collect multi-source evidence, AI-assisted analytics estimate exploitation likelihood and detect anomalies, a graph-aware aggregation layer propagates risk across the service dependency graph, and an integration layer feeds a continuous per-service risk score into the zero-trust policy engine. An illustrative scenario shows how a contextually significant but low-severity vulnerability would be surfaced and contained. The contribution is conceptual rather than experimentally validated; the paper closes by setting out the open challenges of data, robustness, performance, consistency, explainability, and evaluation that lie between the framework and a deployable system.
Keywords: Zero trust architecture; Microservice security; Vulnerability management; Vulnerability prioritisation; Machine learning for security; Cloud-native security; Attack graphs; Continuous security monitoring

1 INTRODUCTION

1.1 The Expanding Attack Surface of Cloud-Native Systems

Enterprise software has moved decisively toward the microservice style, in which an application is built from many small, independently deployable services rather than one large program [1]. The benefits are real: teams ship faster, scale parts of a system independently, and adopt different technologies where each fits. The security consequences are equally real. A systematic view of the field shows that decomposition turns a single trusted application into a sprawling distributed system whose parts communicate over the network [2]. Every internal function call that becomes a network request is a new interface that can be attacked, and the surrounding cloud-native machinery of containers, orchestration, and automated pipelines makes the resulting attack surface large, fast-changing, and hard to enumerate at any one moment.

1.2 The Risk-Assessment Problem

Two trends collide on this surface. The first is the sheer volume of vulnerabilities. Organisations track far more known flaws than they can ever remediate, and deciding which few to address first has become the central problem of vulnerability management [3]. The second is that the tools used to make that decision are poorly matched to distributed systems. The dominant severity-scoring practice rates a vulnerability by its intrinsic technical characteristics, and a prominent critique warns that this severity is routinely and wrongly treated as a measure of risk, even though it ignores whether a flaw is being exploited and ignores the context of the system it sits in [4]. In a microservice deployment, context is everything: the same library flaw is trivial in an isolated service and dangerous in one that sits on a path to sensitive data. Conventional risk assessment compounds the mismatch in three further ways. It is centralised, computing a single global view that lags the rapid change of cloud-native systems. It is static, running as a periodic scan rather than tracking a moving target. And it is disconnected from enforcement, producing reports for human analysts rather than signals that the access-control system can act on.

1.3 The Opportunity: Zero Trust and Artificial Intelligence

Two developments make a better approach possible. The first is the zero-trust security model, which discards the notion of a trusted internal network and instead verifies every request against dynamic policy before granting access [5]. Zero

trust is a natural fit for microservices, because it treats each service-to-service call as untrusted, but realising it requires a continuous supply of trust and risk signals that current systems do not provide. The second development is the maturation of artificial intelligence for security analytics, which can detect vulnerabilities at scale, predict which are likely to be exploited, and reason over the relationships among components [6]. Brought together, these developments suggest a different kind of risk assessment: one that is continuous rather than periodic, distributed rather than centralised, context- and graph-aware rather than severity-only, AI-assisted rather than manual, and wired into zero-trust enforcement rather than filed for review. No existing body of work, however, combines these properties.

1.4 Contributions and Structure

This paper develops that combination as a conceptual framework. It is explicitly a conceptual contribution: a synthesis of four research literatures into a coherent design, not an implemented and evaluated system. The paper makes four contributions. First, it synthesises four normally separate fields, namely zero-trust architecture, microservice and cloud-native security, vulnerability analytics, and software risk management, and shows precisely where they fail to meet (Figure 1).

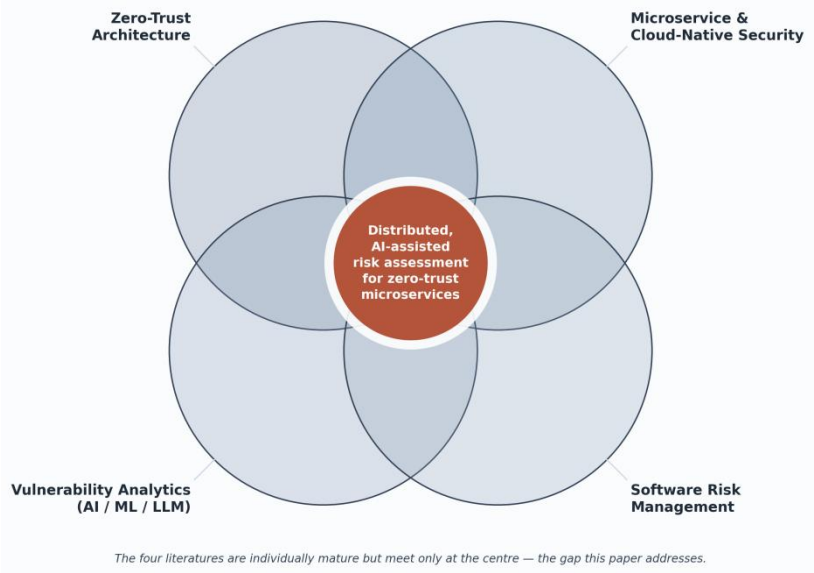

Figure 1 The Four Research Literatures the Paper Draws on are Individually Mature but Meet only at Their Intersection

That intersection, a distributed and AI-assisted risk assessment built for zero-trust microservices, is the gap this paper addresses. Second, it proposes a six-dimensional taxonomy that organises the design space of distributed risk assessment and exposes the region no current approach occupies. Third, it proposes a four-layer conceptual framework for AI-assisted distributed risk assessment that fills that region and connects vulnerability analytics to zero-trust enforcement through a continuous, per-service risk signal. Fourth, it sets out an agenda of the open challenges that stand between the framework and a deployable system.

The remainder of the paper is organised as follows. Section 2 defines the necessary background. Section 3 reviews the four contributing literatures and identifies the gap. Section 4 develops the taxonomy. Section 5 presents the conceptual framework. Section 6 discusses what the framework offers and what it cannot claim. Section 7 sets out open challenges and future directions, and Section 8 concludes.

2 BACKGROUND AND PRELIMINARIES

This section defines the four concepts the paper builds on: microservice and cloud-native architecture, the zero-trust security model, the vulnerability management lifecycle and its scoring systems, and the use of artificial intelligence for security analytics. Readers from outside computer security may find the brief definitions useful, since the argument later in the paper depends on how these ideas interact.

2.1 Microservices and Cloud-Native Architecture

A microservice architecture builds an application as a collection of small, independently deployable services rather than as a single large program. Each service owns one business capability, runs in its own process, and communicates with the others over the network, usually through lightweight web interfaces. A recent survey of the field describes the style as a response to the limits of monolithic systems, offering independent scaling, faster release cycles, and technology diversity across teams [1]. A systematic mapping of the research literature confirms that these benefits come with a cost: the application is now a distributed system, and distribution introduces new failure modes and new security concerns [2].

The shift carries a direct consequence for security. In a monolith, a single trusted boundary surrounds the whole application, and internal calls happen inside one process. In a microservice system, a call that was once an internal function becomes a network request between services, often across containers and hosts. The number of network-exposed interfaces multiplies, and each interface is a potential point of attack. The "cloud-native" label extends this picture with containers, orchestration platforms such as Kubernetes, and automated deployment pipelines. The result is an environment that is elastic and fast to change, but whose attack surface is large, dynamic, and difficult to enumerate at any single moment.

2.2 The Zero-Trust Security Model

Zero trust is a security model that abandons the idea of a trusted internal network. The traditional perimeter model treats everything inside the corporate firewall as trusted and everything outside as hostile. Zero trust rejects this assumption. Its guiding principle, often summarised as "never trust, always verify", holds that no request should be trusted on the basis of its origin alone, whether that origin is inside or outside the network.

The reference definition is the United States National Institute of Standards and Technology publication on zero-trust architecture [5]. It describes an abstract model in which every access request is evaluated before it is granted. A policy engine decides whether to permit a request, a policy administrator establishes or terminates the connection, and a policy enforcement point sits in the data path and applies the decision. The decision draws on multiple signals, including the identity of the requester, the state of the device, and observed behaviour. Access is granted per session and for the minimum necessary scope, and the decision is continually re-evaluated rather than made once at login. This model fits microservices well in principle, because it treats every service-to-service call as an untrusted request that must be verified. Realising it in practice, however, requires a continuous source of trust and risk signals for thousands of interacting services, which is where the present paper concentrates its attention.

2.3 The Vulnerability Management Lifecycle and Its Scoring Systems

A software vulnerability is a flaw that an attacker can exploit to violate the security of a system. Managing vulnerabilities is a continuous lifecycle: discover them, assess their severity, prioritise which to fix first, remediate, and verify. The hard problem is prioritisation. Large organisations face far more known vulnerabilities than they can ever patch, so they must decide which few matter most. A survey of data-driven approaches to vulnerability assessment and prioritisation documents the scale of this problem and the growing use of automated methods to address it [3].

Three scoring systems frame current practice. The Common Vulnerability Scoring System, or CVSS, assigns a severity number to each vulnerability from its technical characteristics. It is widely used but widely criticised. A pointed commentary argues that its severity score is too often mistaken for a measure of risk, when it captures only intrinsic technical severity and ignores whether a flaw is actually being exploited [4]. The Exploit Prediction Scoring System, or EPSS, was proposed to fill that gap by estimating the probability that a given vulnerability will be exploited in the wild [7]. Alongside these, exploited-vulnerability catalogues record flaws that are known to be under active attack. The common thread is that severity alone is a poor guide to action: useful prioritisation must combine technical severity with exploitation likelihood and with the context of the specific system at risk. That contextual dimension is precisely what a distributed microservice environment makes difficult, because the same vulnerability poses different risks in different services.

2.4 Artificial Intelligence for Security Analytics

The final ingredient is the use of machine learning and related techniques to analyse security data. A wide-ranging survey of machine learning in cybersecurity catalogues applications across intrusion detection, malware classification, and vulnerability analysis, and notes both the promise of these methods and their fragility under adversarial conditions [6]. For the purposes of this paper, the relevant capabilities are three. Learning models can help detect and classify vulnerabilities in code at scale. They can help predict which vulnerabilities are likely to be exploited, refining the prioritisation problem described above. And they can model relationships across a system, for instance by reasoning over the graph of services and their dependencies to estimate how risk propagates.

These capabilities are not a panacea. Models require data, they can be wrong, and they can be attacked. The argument of this paper is not that artificial intelligence solves distributed risk assessment. It is that artificial intelligence supplies analytical capabilities a continuous, system-wide risk view requires, capabilities that manual methods cannot provide at the speed and scale of cloud-native systems. With these four concepts in place, the next section reviews how the existing literature has addressed them, and where the gaps lie.

3 RELATED WORK

The problem this paper addresses sits at the meeting point of four research areas: zero-trust architecture, microservice and cloud-native security, vulnerability analytics, and risk management in modern software pipelines. Each area is active and well developed on its own. This section reviews the four in turn and then shows that, despite their maturity, no existing work joins them into a single account of distributed, AI-assisted risk assessment for zero-trust microservices.

The literature reviewed here was gathered through a structured search of the OpenAlex database and verified for authenticity, as summarised in Figure 2.

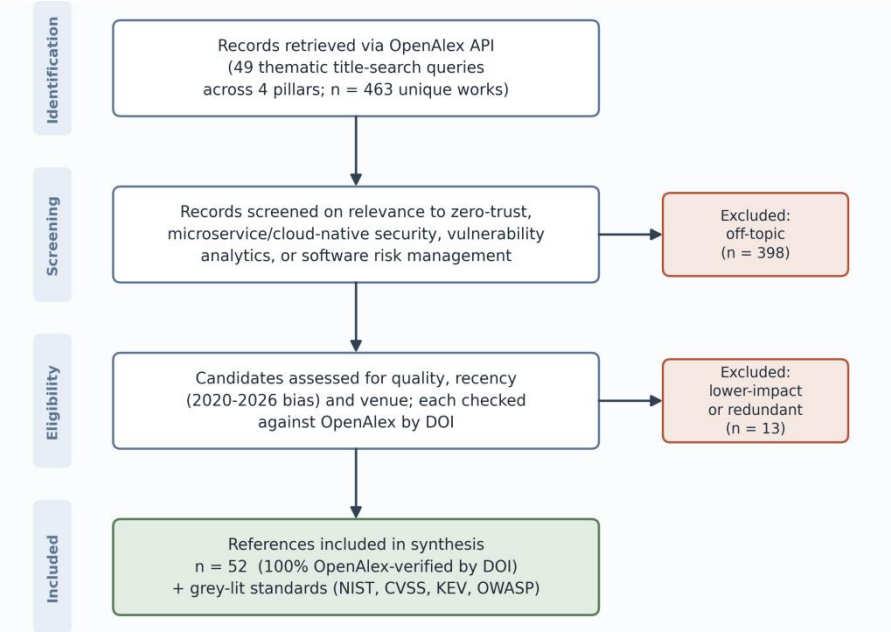

Figure 2 Literature Identification and Verification Flow

Note: Forty-nine thematic title-search queries across the four pillars returned 463 unique works; after relevance, recency, and quality screening, 52 references were retained and verified in full against OpenAlex by DOI, supplemented by grey-literature standards.

3.1 Zero-Trust Architecture and Microservices

Zero trust has moved from a slogan to a substantial research field within a few years. Several recent surveys map its principles and open problems. A broad survey of zero-trust architecture sets out its core tenets and reviews enabling technologies [8], while a parallel survey concentrates on challenges and future trends [9]. A multivocal literature review, which combines academic and practitioner sources, is especially useful because it documents the gap between the model's promise and its deployed reality [10]. Other reviews examine the practical difficulty of migrating existing systems to a zero-trust posture and compare zero-trust approaches specifically within cloud computing [11-12]; a further comparative review analyses competing zero-trust network models [13].

A second strand applies the model to concrete settings. Work on smart healthcare demonstrates a zero-trust protection system for 5G networks [14], and a dynamic access-control system shows how authorisation decisions can be made continuously rather than once per session [15]. Most relevant to the present paper, machine learning has been proposed as the engine that supplies the continuous trust signals a zero-trust system needs. An analysis of intelligent zero-trust architecture for 5G and 6G networks argues that learning models are necessary to evaluate trust at the speed these environments demand [16]. The intersection with microservices, however, remains thin. A study of insider threats in microservice-based smart-grid systems is one of the few works to place zero trust and microservices in the same frame [17]. The literature establishes that zero trust needs continuous, data-driven trust evaluation, but it does not specify how that evaluation should incorporate the vulnerability risk of the individual services being protected.

3.2 Microservice and Cloud-Native Security

Research on securing microservices is similarly mature. Early work catalogued the security challenges that the architectural style introduces, noting that decomposition multiplies the number of trust boundaries [18]. Systematic studies have since consolidated the field. A systematic mapping study of microservice security organises the proposed solutions and identifies recurring themes [19], and two multivocal reviews survey, respectively, the security mechanisms used in practice and the recurring "smells" and refactorings that affect microservice security [20-21]. A broader treatment of security in microservice architectures complements these by surveying patterns and countermeasures [22]. One study addresses the attack surface directly, proposing techniques to seal the interfaces that microservice decomposition exposes [23].

Because microservices are typically packaged in containers and orchestrated by platforms such as Kubernetes, container and orchestration security form an adjacent and essential literature. A widely cited treatment lays out the issues and challenges of container security [24], and a survey of the container technology landscape covers orchestration and isolation [25]. Empirical work has shown that real-world Kubernetes configurations are frequently misconfigured in ways that weaken security [26], and a complementary study explains the security implications of Kubernetes networking [27]. A recent survey consolidates security across cloud-native services as a whole [28]. The service mesh, a dedicated infrastructure layer for service-to-service communication, has become a focal point. A survey describes its

state of the art [29], and two national standards documents specify how to build secure microservice applications using a service mesh and how to implement attribute-based access control through it [30-31]. Access control for microservices is itself an active topic, with proposals for policy-based control across federated cloud and edge deployments and for token-based access control methods [32-33]. This body of work secures the components and their communication, but it treats vulnerability and risk as background conditions rather than as quantities to be continuously measured and acted upon.

3.3 Vulnerability Analytics

The third area concerns the detection, assessment, and prioritisation of vulnerabilities, increasingly with the help of machine learning. Deep-learning vulnerability detection matured rapidly after early systems demonstrated that neural models could find flaws in source code at scale [34]. Graph-based methods proved a natural fit, because code has graph structure. A learning system that identifies vulnerabilities by reasoning over program-semantic graphs became an influential reference point [35]. Later work built bidirectional graph neural networks for the same purpose and combined graph-based learning with automated data collection [36-37]. A taxonomy of software-vulnerability detection consolidates the machine-learning approaches and their datasets [38], and a detection system trained on a natural codebase illustrates the move toward realistic data [39].

The most recent shift is toward large language models. A study of zero-shot vulnerability repair with large language models reports both the promise and the present limits of the approach [40], and an emerging-results paper examines large language models for vulnerability detection directly [41]. A broad survey of the security and privacy of large language models situates these efforts within the wider question of how such models can both aid and endanger security [42]. Beyond detection, a separate line of work targets assessment and prioritisation. Methods have been proposed to predict severity scores from textual vulnerability descriptions and to build frameworks for detecting and prioritising zero-day vulnerabilities [43-44]. Attack-graph generation, which models how an attacker could chain vulnerabilities into a path through a system, has also been approached with machine learning [45]. These techniques are powerful, but they operate largely at the level of individual code units or isolated vulnerabilities. They do not, on their own, produce a system-level risk picture for a running microservice deployment.

3.4 Risk Management, DevSecOps, and the Software Supply Chain

The fourth area concerns how security is managed across the modern software lifecycle. DevSecOps, the integration of security into continuous development and operations, has been studied through a systematic review of its adoption challenges and a decision-making framework for organisations adopting it [46-47]. One contribution is particularly close in spirit to this paper: a conceptual model for automated, continuous security over the cloud, which demonstrates that a security architecture can be specified at the conceptual level and reasoned about before implementation [48]. That precedent supports the methodological stance taken here.

A related concern is the software supply chain, the chain of dependencies and build steps through which vulnerabilities enter a system indirectly. Observations from industry and government organisations identify the leading challenges in securing it [49], and a systematisation-of-knowledge study establishes secure design properties for supply-chain security [50]. The software bill of materials, an inventory of the components in a piece of software, has emerged as a key instrument, and an empirical study assesses where the practice stands and what remains to be done [51]. Finally, runtime observability feeds risk assessment: an unsupervised method detects anomalies in the execution traces of microservices, showing that the behaviour of a running system is itself a source of risk signal [52]. This literature manages security as a process and tracks where risk originates, yet it stops short of a continuous, quantitative, per-service risk score that a zero-trust policy engine could consume directly.

3.5 Synthesis and the Research Gap

Read together, the four areas leave a specific gap. Zero-trust research calls for continuous trust evaluation but does not say how vulnerability risk should inform it. Microservice security research hardens components and communication but treats risk as a static background. Vulnerability analytics produces sophisticated per-vulnerability and per-file judgements but not a system-level, continuously updated risk view. Risk-management research organises the process and locates the sources of risk without delivering the live, per-service score the other three areas need. No existing work combines them. The contribution of this paper is to do so conceptually: to define a taxonomy for distributed risk assessment in zero-trust microservices and to propose a framework in which AI-assisted vulnerability analytics produces the continuous, per-service risk signals that a zero-trust architecture can act upon. The next section develops the taxonomy.

4 A TAXONOMY OF DISTRIBUTED RISK ASSESSMENT FOR ZERO-TRUST MICROSERVICES

To reason about how risk assessment can be made distributed, continuous, and AI-assisted, the design space must first be organised. This section proposes a taxonomy with six dimensions. Each dimension is a question that any risk-assessment design must answer, and the values along each dimension describe the options the literature has explored. The taxonomy serves two purposes: it locates existing techniques within a common map, and it exposes the

combinations that no current approach occupies. These unoccupied combinations motivate the framework proposed in Section 5. Figure 3 presents the taxonomy and marks, on each dimension, the most advanced value that the proposed framework targets.

Figure 3 A Six-Dimension Taxonomy of Distributed Risk Assessment for Zero-Trust Microservices

Note: The accented value on each dimension denotes the most advanced option; the framework of Section 5 targets their combination, a region no current approach occupies.

4.1 Dimension 1: Risk Signal Source

The first question is what evidence a risk assessment draws on. The simplest source is the static catalogue of known vulnerabilities in a service and its dependencies, identified by software composition analysis and recorded in a software bill of materials [51]. A second source is exploitation intelligence, which records whether and how readily a vulnerability is being attacked in the wild, as captured by exploit-prediction scores and exploited-vulnerability catalogues [7]. A third source is the running behaviour of the system, observed through execution traces and logs, from which anomalies can be detected [52]. A fourth is configuration and topology, including network policy and the service-to-service call graph. Approaches differ sharply in how many of these sources they combine. Most current tools rely on a single source, typically the static vulnerability catalogue, and therefore see only part of the risk picture.

4.2 Dimension 2: Assessment Granularity

The second question is the unit at which risk is assessed. Vulnerability-detection research generally operates at the finest granularity, judging individual code units or files [34,39]. Severity and exploitability scores attach to individual vulnerabilities [7,43]. Microservice security, by contrast, naturally concerns coarser units: the service, the container, the API endpoint, and the workload. The coarsest granularity is the whole system, viewed as a graph of interacting services. Attack-graph methods reason at this level by modelling how an attacker could chain steps across components [45]. A central observation is that risk does not compose simply across granularities: a low-severity flaw in a highly connected, internet-facing service can pose more system risk than a high-severity flaw in an isolated one. A distributed assessment must therefore work at several granularities at once and relate them.

4.3 Dimension 3: Analytical Technique

The third question is how the evidence is turned into a judgement. The options span a spectrum of sophistication. The baseline is rule-based scoring, in which fixed formulas convert vulnerability attributes into a severity number, as in conventional severity scoring [4]. Above this sit classical machine-learning models that predict exploitation likelihood or severity from features and text [43]. Deep-learning models, particularly graph neural networks, reason over the structural relationships in code and systems [35-36], and large language models have begun to detect, explain, and repair vulnerabilities from natural-language and code context [40-41]. Unsupervised methods detect anomalies in runtime behaviour without labelled data [52]. Each technique suits a different signal source and granularity, and a complete assessment will combine several rather than rely on one.

4.4 Dimension 4: Risk Model

The fourth question is what notion of risk the output represents. The weakest model equates risk with intrinsic technical severity, the position that severity scoring encourages and that its critics warn against [4]. A stronger model multiplies severity by exploitation likelihood, the contribution of exploit-prediction scoring [7]. A stronger model still adds context: the business criticality of the affected service, its exposure, and its position in the system. The most complete model treats risk as a property of the system graph, in which the risk of one service depends on the risk of its neighbours, so that risk propagates along the paths an attacker could traverse [45]. Moving from severity to context to propagation is the direction in which the literature is heading, but the propagation view remains largely unrealised for live microservice systems.

4.5 Dimension 5: Zero-Trust Integration Point

The fifth question is how the risk judgement connects to enforcement. In most current practice the connection is weak: risk scores are reported to human analysts who decide what to patch, and the assessment never touches the access-control path. A zero-trust architecture offers a tighter coupling, because its policy engine already evaluates every request against multiple signals before granting access [5]. Risk can enter at three points. It can be one input among many to the policy engine's decision, so that a request to or from a high-risk service is scrutinised more heavily. It can shape the policy that the policy administrator installs. And it can trigger continuous re-evaluation, so that a newly discovered vulnerability immediately changes the trust posture of the affected service. Dynamic, continuously re-evaluated access control has been demonstrated in zero-trust systems [15], but feeding live vulnerability risk into that decision is not yet established practice.

4.6 Dimension 6: Temporality and Distribution

The sixth question combines two related properties: when the assessment runs, and where it runs. On timing, approaches range from one-shot static scans, through periodic re-scans, to continuous assessment that updates as the system and the threat landscape change. Continuous security has been argued for at the level of development and operations pipelines [46,48], but continuous risk scoring of running services is rarer. On location, an assessment can be centralised, computing a single global view from collected data, or distributed, with services or local agents contributing to and consuming the assessment near where they run. Centralised assessment is simpler but scales poorly and lags the rapid change of cloud-native systems; distributed assessment matches the architecture of the systems it protects but raises questions of consistency and coordination that remain open.

4.7 Summary: The Unoccupied Region

Placing representative approaches on these six dimensions reveals a clear pattern, summarised in Table 1. The field is rich at the extremes of single dimensions: fine-grained vulnerability detection, sophisticated learning techniques, mature zero-trust models, and continuous development pipelines all exist. Each approach, however, occupies only a corner of the design space. Severity and exploit scoring operate per vulnerability with no system context; learning-based detection works at the level of code units; zero-trust architecture evaluates requests but is blind to vulnerability risk; and attack-graph analysis reaches the system level but runs statically and offline.

Table 1 Representative Approaches Positioned on the Six Taxonomy Dimensions

Approach	Signal source	Granularity	Technique	Risk model	ZT integration	Temporality / distribution
Severity scoring (CVSS) [4]	static CVE	vulnerability	rule-based	severity only	none	static, centralised
Exploit prediction (EPSS) [7]	exploit intel.	vulnerability	classical ML	+ exploitability	none	periodic, centralised
ML/DL vulnerability detection [34], [35], [36]	static code	code unit	deep / graph NN	detection only	none	static, centralised
Zero-trust architecture [5], [8]	identity, device	request / session	rule + policy	trust, not vuln.	core (is the engine)	continuous, centralised
DevSecOps continuous security [46], [48]	pipeline signals	build / service	rule-based	process risk	weak	continuous, centralised
Attack-graph analysis [45]	topology + CVE	system graph	graph / ML	graph propagation	none	static, centralised
Microservice security reviews [19], [20], [21]	mixed	service	qualitative	no live score	varies	static
This framework	all four sources	service ↔ graph	AI-assisted (multiple)	graph propagation	policy-engine input + re-evaluation	continuous, distributed

Note: Each prior approach occupies only part of the space; the proposed framework targets the full combination.

What no current approach provides is the combination that distributed microservice security demands: multiple signal sources, multiple granularities related through the service graph, AI-assisted analysis, a propagation-aware risk model, tight integration with the zero-trust policy engine, and continuous, distributed operation. That combination is the target of the conceptual framework developed next.

5 A CONCEPTUAL FRAMEWORK FOR AI-ASSISTED DISTRIBUTED RISK ASSESSMENT

This section proposes a conceptual framework that occupies the unoccupied region identified by the taxonomy. The framework is conceptual: it is a reasoned synthesis of the reviewed literature into an architecture and a set of design principles, not an implemented system, and it is not accompanied by experimental evaluation. The purpose is to give the field a coherent target to build toward and to make the design choices explicit enough that others can implement, critique, or refute them. Presenting a security architecture at the conceptual level before implementation has an established precedent in this literature [48].

5.1 Design Principles

Six principles follow directly from the taxonomy and the gap analysis. First, multi-source: the assessment must fuse static vulnerability data, exploitation intelligence, runtime behaviour, and configuration, because no single source captures risk. Second, graph-aware: risk must be computed over the service dependency graph so that exposure and connectivity, not severity alone, shape the result. Third, AI-assisted: analytical tasks beyond human speed and scale, such as exploitation prediction and anomaly detection, must be delegated to learning models. Fourth, continuous: the assessment must update as code, configuration, and threat intelligence change, rather than running as a periodic scan. Fifth, zero-trust-integrated: the risk output must be a consumable signal for the policy engine, not a report filed for human review. Sixth, distributed: assessment logic must run close to the services it evaluates, to match the scale and dynamism of cloud-native deployments. These principles are requirements; the architecture below is one way to satisfy them.

5.2 Architecture and Components

The framework has four layers, shown in Figure 4. The description that follows is deliberately abstract, in the manner of the zero-trust reference model it extends [5].

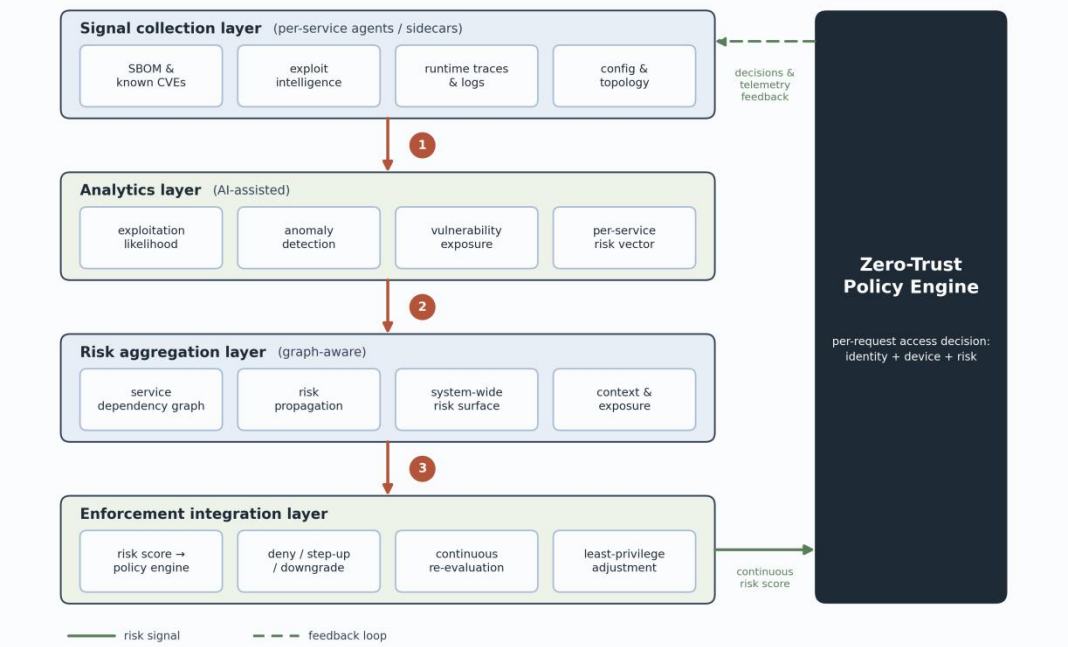

Figure 4 The Four-Layer Conceptual Framework

Note: Per-service agents collect multi-source evidence (top); an AI-assisted analytics layer produces per-service risk vectors; a graph-aware aggregation layer propagates risk across the service dependency graph; and an enforcement integration layer feeds a continuous per-service risk score into the zero-trust policy engine (right), which re-evaluates access per request.

The signal collection layer gathers evidence near each service. A lightweight agent, co-located with the service in the manner of a service-mesh sidecar [29-30], collects four streams. The first is the service's software bill of materials and its known vulnerabilities [51]. The second is relevant exploitation intelligence for those vulnerabilities [7]. The third is runtime telemetry in the form of execution traces and logs [52]. The fourth is local configuration and the identities of the services it communicates with. Collecting near the service keeps the data current and avoids the bottleneck of shipping everything to a central store.

The analytics layer turns evidence into per-service risk estimates using AI-assisted methods. It performs three analytical functions. It estimates each vulnerability's exploitation likelihood, refining catalogue severity with prediction [7,43]. It detects behavioural anomalies in the service's runtime traces that may indicate active compromise [52]. And it characterises the service's intrinsic vulnerability exposure, drawing on the vulnerability-analysis techniques surveyed

earlier [34,38]. The output of this layer is a per-service risk vector rather than a single number, preserving the distinction between latent vulnerability, exploitation likelihood, and observed anomaly.

The risk aggregation layer composes per-service estimates into a system-level view over the service dependency graph. This is the layer that makes the assessment graph-aware. Rather than treating services independently, it propagates risk along the edges of the call graph, so that a service inherits elevated risk from the risky, highly connected neighbours that an attacker could use to reach it. Attack-graph reasoning provides the conceptual basis for this propagation [45]. The result is a continuously updated risk surface over the whole deployment, in which the score of each service reflects both its own state and its position in the system.

The enforcement integration layer connects the risk surface to the zero-trust policy engine. The per-service risk scores become an additional input to the policy engine's access decision, alongside identity and device state [5]. A request involving a service whose risk has risen can be denied, downgraded, or subjected to stronger verification, and because zero-trust decisions are continuously re-evaluated [15], a newly discovered vulnerability can change a service's effective trust within one decision cycle. The framework thus closes the loop from vulnerability discovery to access control without human intervention in the critical path.

5.3 How Artificial Intelligence Assists

The framework assigns artificial intelligence to the tasks where it adds the most value and keeps deterministic logic elsewhere. Three roles are central. In prioritisation, learning models estimate exploitation likelihood and severity from vulnerability descriptions and features, converting an unmanageable list of known flaws into a ranked, context-weighted set [7], [43]. In detection, models identify vulnerabilities in service code and detect anomalies in runtime behaviour, the former drawing on graph and language-model techniques [35,40-41] and the latter on unsupervised trace analysis [52]. In propagation reasoning, graph-based models estimate how risk flows across the service graph, a task whose combinatorial nature defeats manual analysis [45]. Deterministic components handle what they do better than models: collecting signals, applying policy, and enforcing decisions. This division matters, because the surveyed evidence warns that learning models are fallible and can themselves be attacked [6,42]; confining them to analytical roles, with deterministic enforcement downstream, limits the damage a wrong or manipulated inference can do.

5.4 Mapping to Zero-Trust Principles

The framework realises, rather than merely invokes, the tenets of the zero-trust reference model [5]. The tenet that access is granted per request, on the basis of dynamic policy, is served by feeding live per-service risk into each decision. The tenet that the security posture is continuously monitored is served by the continuous, multi-source assessment. The tenet of least privilege is sharpened, because a service's privileges can contract automatically as its risk rises. And the tenet that no asset is inherently trusted is honoured at the granularity of the individual microservice, since every service carries a current, evidence-based risk score rather than a static trust label. In this sense the framework supplies the missing quantity that zero-trust microservice deployments need: a continuous, per-service risk signal derived from vulnerability analytics.

5.5 An Illustrative Scenario

To make the framework concrete, consider a qualitative walkthrough. It is illustrative only; it reports no measurements and claims no evaluated performance. The scenario is depicted in Figure 5. Suppose a public exploit is published for a library used by one internal, normally low-traffic microservice. Under conventional practice, this vulnerability would join a long backlog, ranked by its technical severity, and might wait weeks for a patch. Under the proposed framework, the sequence differs. The signal collection layer registers the new exploitation intelligence against the service's bill of materials. The analytics layer raises the service's exploitation-likelihood estimate. The aggregation layer propagates the elevated risk along the call graph and finds that the affected service, though minor in itself, sits on a path to a customer-data service, so the risk surface around that path rises. The enforcement layer, consuming the updated scores, automatically tightens verification on requests traversing the path, containing the exposure before a patch is available. The same vulnerability that conventional severity ranking would have buried is, in this account, surfaced and contained because its risk was assessed in context, continuously, and in connection with enforcement. Whether a built system would behave this well is an empirical question the framework does not answer; its contribution is to specify a design in which such behaviour becomes possible.

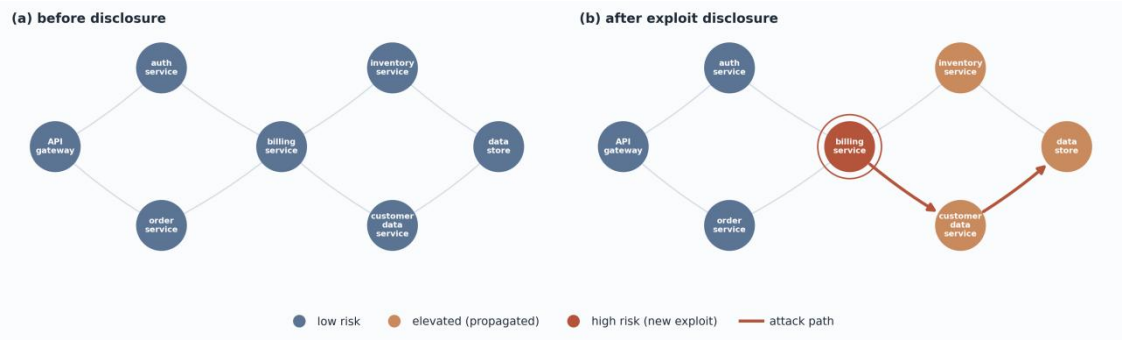

Figure 5 Illustrative Scenario (Schematic, not Measured)

Note: (a) Before disclosure, all services carry low risk. (b) After a public exploit is disclosed for the billing service, the analytics layer raises that service's risk (dashed halo), and the aggregation layer propagates elevated risk along the dependency graph to the customer-data path (bold arrows), so the enforcement layer can tighten verification on that path before a patch is available.

6 DISCUSSION

6.1 What the Framework Offers

The framework's value lies in joining four capabilities that the literature has developed separately. By fusing static vulnerability data, exploitation intelligence, runtime behaviour, and topology, it replaces the single-source view of conventional tools with a fuller picture of risk. By computing risk over the service graph, it corrects the central error that severity scoring encourages, namely treating a flaw's technical severity as its risk regardless of where the flaw sits [4]. By delegating prediction, detection, and propagation reasoning to learning models, it operates at a speed and scale that manual assessment cannot reach [6]. And by feeding a continuous per-service risk score into the zero-trust policy engine, it turns risk assessment from a reporting activity into a control input. The discovery of a vulnerability can then change a service's effective trust within one decision cycle [5,15]. The illustrative scenario in Section 5 shows how these capabilities combine: a flaw that severity ranking would bury is surfaced and contained because its risk is assessed in context, continuously, and in connection with enforcement.

6.2 Deployment Considerations

Several properties of the framework ease its adoption. Because the signal collection agent is designed to sit beside each service in the manner of a service-mesh sidecar [29-30], it can be deployed incrementally, service by service, without rewriting applications. Because the analytics and enforcement layers consume standard artefacts, namely bills of materials, exploitation feeds, traces, and policy decisions, they can build on tools that many organisations already run. The framework is therefore better understood as an integrating architecture than as a replacement for existing investments. Its natural home is an organisation that has already adopted microservices, has begun a zero-trust programme, and practises continuous development and operations, for which the framework supplies the missing connective tissue between vulnerability data and access control [46,48].

6.3 The Convergence-with-Context Argument and Its Limits

The framework rests on a single claim: that risk in a distributed system is a property of context and connectivity, not of isolated vulnerabilities, and that assessing it therefore requires a continuous, graph-aware, AI-assisted, enforcement-connected approach. The reviewed literature supports each component of this claim, but the claim about their combination is, at this stage, an argued position rather than a demonstrated result. This is the honest boundary of a conceptual contribution. The paper does not show that a built system would be accurate, fast enough, or robust; it shows that such a system is coherent, well-motivated by the literature, and specified clearly enough to be built and tested. The history of the field offers a precedent for this mode of contribution, where a conceptual model is articulated and reasoned about before implementation [48]. The value of the framework is to convert a diffuse set of unmet needs into a concrete, falsifiable design.

7 OPEN CHALLENGES AND FUTURE DIRECTIONS

The distance between the proposed framework and a deployable system is measured by a set of open challenges, each of which is also a research direction.

7.1 Data Availability and Quality

The analytics layer depends on data: labelled vulnerabilities, reliable exploitation intelligence, and representative runtime traces. The learning literature repeatedly shows that model quality is bounded by data quality [38], [39], and exploitation intelligence in particular is incomplete and biased toward what is observed. Building and maintaining

trustworthy data pipelines for live microservice systems, and quantifying the uncertainty that imperfect data introduces into risk scores, is a primary challenge.

7.2 Robustness and Adversarial Manipulation

Learning models can be wrong, and they can be deliberately deceived. The surveyed evidence stresses that security models are themselves attack targets, vulnerable to adversarial and poisoning manipulation [6,42]. A risk-assessment system that drives access control is an especially attractive target, since an attacker who can suppress a service's risk score can lower its defences. Confining models to analytical roles with deterministic enforcement downstream, as the framework does, limits but does not eliminate this exposure. Making AI-assisted risk assessment robust against adversaries who understand it is a hard and necessary direction.

7.3 Performance and the Cost of Continuity

Continuous, graph-aware assessment has a cost. Collecting telemetry from every service, running analytics, propagating risk across a large service graph, and feeding scores into every access decision all consume resources and add latency. Service-mesh sidecars already impose measurable overhead, and adding analytics to them compounds it. Determining whether continuous distributed risk assessment can run within acceptable performance budgets, and where to trade timeliness for cost, is an open empirical question that only implementation can answer.

7.4 Consistency in Distributed Assessment

Distributing the assessment to run near services raises the classic difficulties of distributed systems. If each agent holds a partial view, the system must reconcile these into a consistent risk surface despite delays and failures, and the propagation of risk across the graph must converge rather than oscillate. How to coordinate distributed risk assessment without reintroducing the central bottleneck it was meant to avoid is a substantial design problem.

7.5 Explainability and Trust in the Decision

When a risk score tightens or denies access, operators and auditors will need to understand why. Opaque model outputs are poorly suited to security decisions that must be justified and contested. Producing risk assessments that are not only accurate but explainable, so that a denied request can be traced to the vulnerability, exploitation signal, or anomaly that caused it, is essential for the framework to be trusted in practice.

7.6 Evaluation and Standardisation

Finally, the field lacks agreed ways to evaluate a system of this kind. There are no standard benchmarks for distributed, continuous risk assessment, no accepted metrics that capture whether contextual risk scoring outperforms severity ranking in operational terms, and no shared datasets that combine vulnerabilities, topology, and runtime behaviour. Developing such benchmarks, and aligning the framework with emerging standards for zero-trust and software transparency [5,51], would let competing designs be compared and would turn the conceptual framework into a testable engineering programme.

8 CONCLUSION

Microservice and cloud-native architectures have enlarged the software attack surface faster than the methods used to assess its risk have adapted. Conventional vulnerability management remains centralised, static, severity-focused, and disconnected from enforcement, none of which suits a distributed system whose risk depends on context and connectivity. This paper has argued that a different approach is both needed and, given recent advances, possible. It synthesised four separate research literatures, namely zero-trust architecture, microservice and cloud-native security, vulnerability analytics, and software risk management, and showed that none of them, alone, delivers a continuous, system-wide, enforcement-connected risk view. It organised the design space with a six-dimensional taxonomy and identified the combination that no current approach provides. And it proposed a four-layer conceptual framework in which AI-assisted analytics produce a continuous, per-service, graph-aware risk signal that a zero-trust policy engine can act upon directly.

The contribution is conceptual, and its limits should be stated plainly. The framework is reasoned from the literature, not validated by experiment. The open challenges of data, robustness, performance, consistency, explainability, and evaluation stand between it and a deployable system. Its purpose is to give a fragmented field a coherent and falsifiable target. Treating distributed risk assessment as a first-class, continuous, AI-assisted component of zero-trust microservice security, rather than as an afterthought to vulnerability scanning, is the shift this paper advocates, and the agenda above marks the path toward realising it.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

REFERENCES

[1] Velepucha Flores P. A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. IEEE Access, 2023. DOI: 10.1109/ACCESS.2023.3305687.

[2] Di Francesco P, Malavolta I, Lago P, et al. Architecting with Microservices: A Systematic Mapping Study. Journal of Systems and Software, 2019. DOI: 10.1016/j.jss.2019.01.001.

[3] Le T, Bao L, Lo D, et al. A Survey on Data-Driven Software Vulnerability Assessment and Prioritization. ACM Computing Surveys, 2022. DOI: 10.1145/3529757.

[4] Spring J, Hatleback E, Householder A, et al. Time to Change the CVSS? IEEE Security & Privacy, 2021. DOI: 10.1109/MSEC.2020.3044475.

[5] Rose S, Borchert O, Mitchell S, et al. Zero Trust Architecture (NIST SP 800-207). NIST Special Publication, 2020. DOI: 10.6028/NIST.SP.800-207.

[6] Dasgupta D, Akhtar Z, Sen S. Machine Learning in Cybersecurity: A Comprehensive Survey. The Journal of Defense Modeling and Simulation, 2020. DOI: 10.1177/1548512920951275.

[7] Jacobs J, Romanosky S, Edwards B, et al. Exploit Prediction Scoring System (EPSS). Digital Threats: Research and Practice, 2021. DOI: 10.1145/3436242.

[8] Syed M H, Fernandez E B, Ilyas M, et al. Zero Trust Architecture (ZTA): A Comprehensive Survey. IEEE Access, 2022. DOI: 10.1109/ACCESS.2022.3174679.

[9] He Y, Li Y, Zhang X, et al. A Survey on Zero Trust Architecture: Challenges and Future Trends. Wireless Communications and Mobile Computing, 2022. DOI: 10.1155/2022/6476274.

[10] Buck C, Olenberger C, Schweizer A, et al. Never Trust, Always Verify: A Multivocal Literature Review on Current Knowledge and Research Gaps of Zero-Trust. Computers & Security, 2021. DOI: 10.1016/j.cose.2021.102436.

[11] Teerakanok S, Uehara T, Inomata A. Migrating to Zero Trust Architecture: Reviews and Challenges. Security and Communication Networks, 2021. DOI: 10.1155/2021/9947347.

[12] Sarkar S, Chatterjee S, Misra S, et al. Security of Zero Trust Networks in Cloud Computing: A Comparative Review. Sustainability, 2022. DOI: 10.3390/su141811213.

[13] Dhiman G, Singh S, Alenezi M, et al. A Review and Comparative Analysis of Relevant Approaches of Zero Trust Network Model. Sensors, 2024. DOI: 10.3390/s24041328.

[14] Chen S, Zhang Y, Li Y, et al. A Security Awareness and Protection System for 5G Smart Healthcare Based on Zero-Trust Architecture. IEEE Internet of Things Journal, 2020. DOI: 10.1109/JIOT.2020.3041042.

[15] Yao X, Wang H, Wang P, et al. Dynamic Access Control and Authorization System Based on Zero-Trust Architecture. ACM Conference Proceedings, 2020. DOI: 10.1145/3437802.3437824.

[16] Ramezanpour K, Jagannath J. Intelligent Zero Trust Architecture for 5G/6G Networks: Principles, Challenges, and the Role of Machine Learning. Computer Networks, 2022. DOI: 10.1016/j.comnet.2022.109358.

[17] Stanojević M, Stojanović Z, Tasić N, et al. Fighting Insider Threats with Zero-Trust in Microservice-Based Smart Grid OT Systems. Acta Polytechnica Hungarica, 2023. DOI: 10.12700/APH.20.6.2023.6.13.

[18] Yarygina T, Bagge A H. Overcoming Security Challenges in Microservice Architectures. IEEE SOSE, 2018. DOI: 10.1109/SOSE.2018.00011.

[19] Hannousse A, Yahiouche S. Securing Microservices and Microservice Architectures: A Systematic Mapping Study. Computer Science Review, 2021. DOI: 10.1016/j.cosrev.2021.100415.

[20] Pereira-Vale A, Márquez G, Astudillo H, et al. Security in Microservice-Based Systems: A Multivocal Literature Review. Computers & Security, 2021. DOI: 10.1016/j.cose.2021.102200.

[21] Ponce F, Márquez G, Astudillo H. Smells and Refactorings for Microservices Security: A Multivocal Literature Review. Journal of Systems and Software, 2022. DOI: 10.1016/j.jss.2022.111393.

[22] Mateus-Coelho N, Cruz-Correia R, Adão P. Security in Microservices Architectures. Procedia Computer Science, 2021. DOI: 10.1016/j.procs.2021.01.320.

[23] Chen L, Magdy W, Rahman M A, et al. With Great Abstraction Comes Great Responsibility: Sealing the Microservices Attack Surface. IEEE SecDev, 2019. DOI: 10.1109/SecDev.2019.00027.

[24] Sultan S, Ahmad I, Dimitriou T. Container Security: Issues, Challenges, and the Road Ahead. IEEE Access, 2019. DOI: 10.1109/ACCESS.2019.2911732.

[25] Casalicchio E, Iannucci S. The State-of-the-Art in Container Technologies: Application, Orchestration and Security. Concurrency and Computation: Practice and Experience, 2020. DOI: 10.1002/cpe.5668.

[26] Rahman M A, Williams L, Meneely A, et al. Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study. ACM Conference Proceedings, 2023. DOI: 10.1145/3579639.

[27] Minna S, Vaarandi R, Pezaros D, et al. Understanding the Security Implications of Kubernetes Networking. IEEE Security & Privacy, 2021. DOI: 10.1109/MSEC.2021.3094726.

[28] Theodoropoulos A, Mylonas A, Gritzalis D. Security in Cloud-Native Services: A Survey. Journal of Cybersecurity and Privacy, 2023. DOI: 10.3390/jcp3040034.

[29] Li W, Lemieux Y, Gao J, et al. Service Mesh: Challenges, State of the Art, and Future Research Opportunities. IEEE SOSE, 2019. DOI: 10.1109/SOSE.2019.00026.

[30] Chandramouli R. Building Secure Microservices-Based Applications Using Service-Mesh Architecture (NIST SP 800-204A). NIST Special Publication, 2020. DOI: 10.6028/NIST.SP.800-204A.

[31] Chandramouli R, Butcher Z. Attribute-Based Access Control for Microservices-Based Applications Using a Service Mesh (NIST SP 800-204B). NIST Special Publication, 2021. DOI: 10.6028/NIST.SP.800-204B.
[32] Preuveneers D, Joosen W. Towards Multi-party Policy-based Access Control in Federations of Cloud and Edge Microservices. IEEE EuroS&PW, 2019. DOI: 10.1109/EUROSPW.2019.00010.
[33] Venčkauskas A, Toldinas J, Grigaliūnas Š, et al. Enhancing Microservices Security with Token-Based Access Control Method. Sensors, 2023. DOI: 10.3390/s23063363.
[34] Li Z, Zou D, Xu S, et al. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. NDSS, 2018. DOI: 10.14722/ndss.2018.23158.
[35] Zhou Y, Liu S, Siow J, et al. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. arXiv, 2019. DOI: 10.48550/arXiv.1909.03496.
[36] Cao Y, Liu X, Wen J, et al. BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection. Information and Software Technology, 2021. DOI: 10.1016/j.infsof.2021.106576.
[37] Wang S, Liu T, Tan L, et al. Combining Graph-Based Learning With Automated Data Collection for Code Vulnerability Detection. IEEE TIFS, 2020. DOI: 10.1109/TIFS.2020.3044773.
[38] Hanif M, Maffeis S, Sasse M A, et al. The Rise of Software Vulnerability: Taxonomy of Software Vulnerabilities Detection and Machine Learning Approaches. Journal of Network and Computer Applications, 2021. DOI: 10.1016/j.jnca.2021.103009.
[39] Wartschinski L, Noller Y, Rieck K, et al. VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python. Information and Software Technology, 2022. DOI: 10.1016/j.infsof.2021.106809.
[40] Pearce H, Ahmad B, Tan B, et al. Examining Zero-Shot Vulnerability Repair with Large Language Models. IEEE Symposium on Security and Privacy, 2023. DOI: 10.1109/SP46215.2023.10179420.
[41] Zhou Y, Liu S, Zhang J, et al. Large Language Model for Vulnerability Detection: Emerging Results and Future Directions. ACM Conference Proceedings, 2024. DOI: 10.1145/3639476.3639762.
[42] Yao Y, Xiao N, Liu Z, et al. A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly. High-Confidence Computing, 2024. DOI: 10.1016/j.hcc.2024.100211.
[43] Costa D, Antunes N, Vieira M. Predicting CVSS Metric via Description Interpretation. IEEE Access, 2022. DOI: 10.1109/ACCESS.2022.3179692.
[44] Singh J, Joshi K P. A Framework for Zero-Day Vulnerabilities Detection and Prioritization. Journal of Information Security and Applications, 2019. DOI: 10.1016/j.jisa.2019.03.011.
[45] Koo D, Kim J, Cho S, et al. Attack Graph Generation with Machine Learning for Network Security. Electronics, 2022. DOI: 10.3390/electronics11091332.
[46] Rajapakse R N, Zahedi M, Babar M A. Challenges and Solutions When Adopting DevSecOps: A Systematic Review. Information and Software Technology, 2021. DOI: 10.1016/j.infsof.2021.106700.
[47] Akbar M A, Shameem M, Ahmad S, et al. Toward Successful DevSecOps in Software Development Organizations: A Decision-Making Framework. Information and Software Technology, 2022. DOI: 10.1016/j.infsof.2022.106894.
[48] Kumar R, Goyal R. Modeling Continuous Security: A Conceptual Model for Automated DevSecOps Using Open-Source Software over Cloud (ADOC). Computers & Security, 2020. DOI: 10.1016/j.cose.2020.101967.
[49] Enck W, Williams L. Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations. IEEE Security & Privacy, 2022. DOI: 10.1109/MSEC.2022.3142338.
[50] Okafor C, Munaiah N, Meneely A, et al. SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties. ACM CCS, 2022. DOI: 10.1145/3560835.3564556.
[51] Xia P, Bao L, Lo D, et al. An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead. ICSE, 2023. DOI: 10.1109/ICSE48619.2023.00219.
[52] Liu Y, Wang Y, Li H, et al. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. ISSRE, 2020. DOI: 10.1109/ISSRE5003.2020.00014.