Software Engineering and Artificial Intelligence Systems

Volume: 02 Issue: 01 (2026)

www.seais.org/

Hybrid SAST DAST Prioritization for Cloud Native Payment Service Vulnerabilities at Service Scale

Minho Baek, Harin Son, Asuka Kameda
Faculty of Health and Medical Sciences, University of Copenhagen, Copenhagen, Denmark

Key word:
Cloud-Native Security, Vulnerability Prioritization, Application Security Testing, Microservices Architecture, Financial Technology

A B S T R A C T

The rapid transition of financial infrastructure toward cloud-native architectures has fundamentally transformed the development, deployment, and maintenance of payment services. While microservices, containerization, and continuous integration methodologies offer unprecedented scalability and agility, they also introduce immense security challenges. Traditional security testing paradigms, such as standalone Static Application Security Testing and Dynamic Application Security Testing, frequently generate overwhelming volumes of vulnerability alerts, many of which are false positives or lack actionable context. This issue is particularly acute in large-scale payment services, where the regulatory environment demands rigorous compliance and the operational environment demands high availability. This paper presents a comprehensive framework for a hybrid vulnerability prioritization model that integrates static code analysis with dynamic runtime testing. By correlating the theoretical vulnerabilities identified during the static phase with the exploitable attack surfaces discovered during the dynamic phase, the proposed framework significantly reduces the noise associated with standalone testing methodologies. Furthermore, the methodology incorporates environmental context and business criticality into the risk scoring algorithm, providing security operations teams with a prioritized remediation queue that aligns with organizational risk tolerance. Through extensive theoretical modeling and empirical analysis within a simulated enterprise payment environment, this research demonstrates that the hybrid prioritization approach yields a substantial reduction in false positives and accelerates the mean time to remediation for critical vulnerabilities. The findings offer a scalable, automated solution for integrating robust security testing into the rapid release cycles characteristic of modern financial technology operations.

INTRODUCTION

1.1 The Evolution of Cloud-Native Payment Architectures

The global financial technology sector has experienced a paradigm shift over the past decade, moving away from monolithic application architectures housed in on-premises data centers toward distributed, cloud-native deployments. This transition has been driven by the need for financial institutions and payment processors to rapidly scale their operations, handle massive

transaction volumes during peak periods, and deploy new features with minimal downtime. Cloud-native architectures leverage containerization technologies, orchestration platforms, and microservices design patterns to decompose complex payment processing applications into smaller, independently deployable services. Each microservice typically encapsulates a specific business capability, such as user authentication, transaction routing, fraud detection, or ledger management. While this architectural decomposition provides significant operational benefits, it simultaneously expands the attack surface of the overall system. In a traditional monolithic application, internal function calls are executed within the same memory space, whereas in a microservices architecture, these interactions are replaced by network-based inter-process communication mechanisms, primarily utilizing application programming interfaces over standard web protocols. The proliferation of these internal interfaces necessitates rigorous security controls, as each endpoint represents a potential vector for exploitation [1]. Furthermore, the deployment frequency in cloud-native environments is magnitudes higher than in traditional settings, with continuous integration and continuous deployment pipelines facilitating multiple application updates per day. This accelerated release cadence places immense pressure on security teams to validate the integrity and safety of the software without impeding the development lifecycle, leading to a fundamental tension between operational agility and information security [2].

1.2 Security Challenges and the Alert Fatigue Phenomenon

Securing cloud-native payment services requires a multifaceted approach to vulnerability management. Payment systems handle highly sensitive financial data, including primary account numbers, personal identifiable information, and cryptographic keys. Consequently, they are subject to stringent regulatory frameworks and industry standards, such as the Payment Card Industry Data Security Standard. To achieve compliance and protect customer assets, organizations deploy various automated application security testing tools throughout the development lifecycle [3]. Static Application Security Testing tools analyze the source code, bytecode, or binary code of an application at rest to identify coding flaws, insecure dependencies, and hardcoded secrets. Dynamic Application Security Testing tools, on the other hand, interact with the running application from the outside, sending crafted payloads to identify runtime vulnerabilities such as injection flaws, authentication bypasses, and server misconfigurations. While both methodologies are essential for a comprehensive security posture, their independent application in large-scale environments often leads to systemic inefficiencies. Static analysis tools are notorious for generating high rates of false positives because they lack visibility into the runtime environment and the compensating controls that may mitigate a theoretical vulnerability [4]. Dynamic analysis tools produce fewer false positives but often struggle to achieve deep code coverage and cannot pinpoint the exact line of code responsible for an identified flaw. When these tools are deployed in an enterprise setting with thousands of microservices, they collectively generate tens of thousands of security alerts. Security analysts are subsequently tasked with manually reviewing, verifying, and prioritizing these alerts, a process that is both time-consuming and prone to human error. This relentless influx of unactionable data culminates in alert fatigue, a psychological and operational phenomenon where critical vulnerabilities are overlooked amid the noise of trivial or erroneous warnings.

1.3 Rationale for a Hybrid Prioritization Framework

To address the limitations of standalone testing methodologies and mitigate the risks associated with alert fatigue, there is a critical need for an automated, hybrid approach to vulnerability prioritization. By intelligently correlating the findings from static and dynamic analysis, organizations can achieve a higher degree of confidence in the validity and exploitability of identified flaws. A hybrid framework posits that a vulnerability is of paramount concern only if it is both present in the underlying codebase and accessible via a reachable execution path in the deployed environment. Moreover, effective prioritization cannot rely solely on the technical severity of a vulnerability, as defined by standard scoring systems. It must also incorporate contextual factors, such as the business criticality of the affected microservice, the sensitivity

of the data processed by that service, and the architectural deployment configuration [5]. For example, a severe injection vulnerability in an isolated, internal reporting tool poses a significantly lower risk than a moderate vulnerability in an internet-facing payment gateway. This paper proposes a comprehensive framework designed specifically for cloud-native payment services operating at scale. The framework ingests telemetry from multiple security tooling streams, normalizes the vulnerability data, and applies a multi-dimensional scoring algorithm to generate a contextualized priority queue. By prioritizing vulnerabilities based on corroborated evidence and environmental context, the proposed system enables security operations teams to allocate their limited resources toward remediating the most critical threats, thereby enhancing the overall resilience of the financial infrastructure.

2. Literature Review and Background

2.1 Static Application Security Testing Paradigms

Static Application Security Testing represents one of the foundational pillars of modern software security assurance. By examining the application codebase without executing the program, static analysis tools attempt to identify security weaknesses early in the software development lifecycle. The fundamental mechanics of static testing involve translating the source code into an intermediate representation, such as an abstract syntax tree or a control flow graph, which can be computationally analyzed [6]. Taint analysis is a primary technique utilized within this domain. It tracks the flow of untrusted user input from potential entry points to sensitive execution sinks. If the analysis determines that the input can reach a sink without undergoing appropriate sanitization or validation, a vulnerability is flagged. Despite its historical success, static analysis faces significant challenges when applied to modern, distributed payment architectures. The complexity of modern programming languages, the heavy reliance on external frameworks, and the use of reflection or dynamic code loading often obscure the true execution path, leading static tools to make conservative assumptions. This conservatism inherently produces a high volume of false positives [7]. Furthermore, static analysis tools are entirely blind to the deployment context. They cannot determine whether a microservice will be deployed behind a robust web application firewall, whether network policies will restrict its communication to a trusted subnet, or whether identity and access management controls will limit its execution privileges [8]. Consequently, the severity ratings assigned by static testing tools represent a worst-case theoretical scenario rather than the actual risk posed in the production environment.

2.2 Dynamic Application Security Testing Methodologies

In contrast to static analysis, Dynamic Application Security Testing evaluates applications in their running state. Dynamic tools simulate the actions of a malicious external actor by crawling the application interfaces and systematically injecting varied attack payloads to uncover vulnerabilities. This methodology is highly effective at identifying misconfigurations, flawed authentication mechanisms, state management issues, and vulnerabilities that only manifest during runtime execution [9]. For cloud-native payment services, dynamic testing is critical for validating the security of application programming interfaces, which often serve as the primary communication mechanism between frontend clients and backend payment processing engines. Dynamic testing utilizes techniques such as protocol fuzzing, parameter tampering, and boundary condition testing to elicit unexpected behavior from the application. However, dynamic testing is not without its limitations. It is inherently a black-box testing methodology, meaning it operates without visibility into the internal structure of the application. Therefore, it may fail to discover vulnerabilities hidden deep within the application logic or along execution paths that are difficult to trigger through standard external interfaces [10]. Additionally, setting up effective dynamic testing environments that accurately mirror the complexity of a production payment system is a significant operational challenge. Automated dynamic scans can also introduce instability into the testing environment, generating superfluous data in application logs or inadvertently triggering disruptive state changes [11]. The inability of dynamic testing to provide exact remediation guidance, such as the specific

file and line number containing the flaw, further complicates the vulnerability resolution process for development teams.

2.3 Current Approaches to Vulnerability Prioritization

The industry standard for communicating the characteristics and severity of software vulnerabilities is the Common Vulnerability Scoring System. This framework provides a standardized method for capturing the principal technical characteristics of a vulnerability and producing a numerical score reflecting its severity. The score is calculated based on metrics such as the attack vector, attack complexity, privileges required, user interaction, and the potential impact on confidentiality, integrity, and availability [12]. While the scoring system is universally adopted, it has been widely criticized for its limitations in enterprise vulnerability management. Its primary shortcoming is that it measures technical severity in a vacuum, without accounting for the specific environmental context or the business value of the affected asset. A vulnerability with a high base score may pose negligible risk if it is located on an isolated, non-critical server. Conversely, a medium-severity vulnerability on a critical payment processing node may require immediate remediation. Recognizing these limitations, researchers and practitioners have proposed various enhanced prioritization models. Some models integrate threat intelligence feeds to prioritize vulnerabilities that are actively being exploited by threat actors in the wild [13]. Others leverage machine learning algorithms to predict the likelihood of exploitation based on historical data and the textual description of the vulnerability. However, many of these approaches remain theoretical or require extensive manual tuning, making them difficult to scale across highly dynamic, cloud-native environments characterized by transient infrastructure and microservices [14].

2.4 Research Gaps in Cloud-Native Payment Contexts

Despite the advancements in individual security testing methodologies and prioritization frameworks, significant research gaps remain regarding their application in large-scale, cloud-native payment systems. Most existing hybrid security models focus on traditional web applications and do not adequately address the complexities of microservices architectures. In a microservices environment, a single business transaction may traverse a dozen different services, each written in a different programming language and managed by a different development team [15]. A vulnerability in one service could potentially be exploited to compromise downstream services, necessitating a prioritization approach that considers the aggregate risk across the entire service graph. Furthermore, payment systems operate under unique constraints, where the integrity and non-repudiation of transactions are just as critical as the confidentiality of the data. Existing prioritization algorithms often fail to appropriately weight these domain-specific requirements. There is a lack of comprehensive frameworks that automatically correlate the theoretical findings from static analysis pipelines with the empirical evidence from dynamic testing, while simultaneously overlaying the architectural context and business criticality of financial applications [16]. This research aims to bridge this gap by proposing a deterministic, automated methodology for hybrid vulnerability prioritization that is specifically tailored to the operational realities of massive-scale payment processors.

3. Methodology and Analytical Framework

3.1 System Architecture of the Hybrid Approach

The proposed hybrid prioritization framework is designed to operate seamlessly within a standard continuous integration and continuous deployment pipeline, providing asynchronous security analysis without blocking the rapid delivery of software updates. The architecture comprises four distinct logical layers: the ingestion layer, the normalization layer, the correlation engine, and the contextual scoring module. The ingestion layer acts as a centralized repository, receiving disparate data streams from multiple security testing tools. During the build phase of the software lifecycle, static analysis agents scan the source code of individual microservices, generating raw vulnerability reports. Concurrently, software composition analysis tools inspect the dependency tree for known vulnerable third-party libraries. Once the application is deployed to an ephemeral staging or integration environment, dynamic testing agents initiate automated security assessments against the exposed interfaces. The raw outputs

from these varied tools are typically formatted according to proprietary schemas, presenting a significant barrier to unified analysis. The normalization layer addresses this challenge by parsing the disparate data formats and mapping them onto a standardized, extensible schema. This schema captures essential attributes such as the vulnerability type, the affected component, the identified execution path or endpoint, the initial severity rating, and the specific tool that generated the finding. Following normalization, the data is forwarded to the correlation engine. This component serves as the core analytical brain of the framework, tasked with identifying intersecting findings between the static and dynamic datasets. The correlation process relies on deterministic matching rules that link theoretical vulnerabilities to exploitable endpoints. For example, if a static analysis tool identifies a potential structured query language injection vulnerability in a specific database access method, and a dynamic analysis tool subsequently successfully executes an injection payload against an interface that routes to that method, the correlation engine links the two findings. This corroboration fundamentally transforms the nature of the alert, elevating it from a potential risk to a confirmed, exploitable vulnerability.

3.2 Vulnerability Aggregation and Correlation Mechanics

The correlation engine utilizes a graph-based approach to map the relationships between identified vulnerabilities and the architectural topology of the payment system. In a microservices architecture, services communicate over defined network pathways. By integrating the framework with the service mesh or the application programming interface gateway, the system constructs a dependency graph that illustrates the flow of data between services. When static analysis identifies a vulnerability, the framework maps it to the specific microservice node within the graph. Dynamic findings are similarly mapped based on the endpoint targeted during the test. The correlation algorithm evaluates the proximity and relationship between these mapped findings. Exact correlation occurs when static and dynamic findings pinpoint the same vulnerability category on the same specific application component. However, the system also supports fuzzy correlation, which identifies probabilistic relationships between findings that may not align perfectly due to limitations in the testing tools [17]. For instance, a dynamic tool might flag a generic server error in response to a malformed payload, while a static tool identifies an unhandled exception in the routing module. The fuzzy correlation logic links these findings as a high-probability match, requiring further investigation but assigning a higher confidence score than standalone alerts. This multi-layered correlation strategy ensures that the system can operate effectively even when the underlying security tools provide incomplete or asymmetric data.

Table 1: Vulnerability Attribute Normalization Parameters

Source Domain	Raw Output Attribute	Standardized Schema Mapping	Description of Normalized Value
Static Analysis	Issue Type	Vulnerability Classification	Standardized weakness enumeration category
Static Analysis	File Path and Line	Component Location	Uniform resource identifier or module path
Dynamic Analysis	Affected Uniform Resource Locator	Attack Surface Endpoint	Network accessible interface identifier
Dynamic Analysis	Payload Response	Exploitability Evidence	Extracted text confirming payload execution
Infrastructure	Container Image	Execution Environment	Base image and orchestration metadata

3.3 Contextual Risk Scoring Mechanism

While correlating static and dynamic findings significantly increases confidence in the validity of a vulnerability, prioritization requires the integration of environmental and business context.

The contextual scoring module adjusts the baseline severity of a corroborated vulnerability by applying a series of multidimensional modifiers. The baseline score is initially derived from the standard vulnerability scoring framework, representing the intrinsic technical severity. The first modifier applied is the asset criticality index. In a payment processing environment, microservices are categorized based on their function. A service that processes real-time transaction authorizations is assigned a higher criticality index than a service responsible for generating end-of-month statements. The second modifier is the environmental exposure factor. This factor evaluates the network positioning of the vulnerable service. A microservice that is directly accessible from the public internet receives a high exposure penalty, whereas a service buried deep within a private subnet, accessible only by specific internal peers, receives a mitigation discount. The third modifier is the data sensitivity classification, which assesses the type of information handled by the vulnerable component. Services interacting with primary account numbers, cryptographic keys, or personal identifiable information are penalized more heavily than those handling generic telemetry data. The final contextual score is calculated through a weighted aggregation of the baseline score and these specific modifiers [18]. This calculation produces a dynamic risk rating that accurately reflects the potential impact of the vulnerability on the specific organization. By utilizing this contextual scoring mechanism, security teams can dynamically sort the remediation queue, ensuring that developers are focused on the flaws that present the most imminent and severe threat to the payment infrastructure, rather than those that simply triggered a high alert in a generic testing tool.

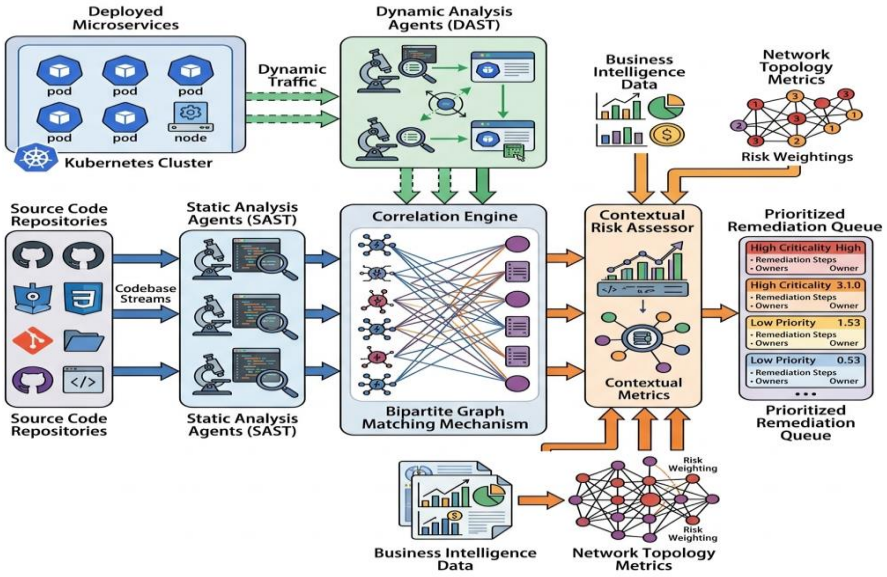


Figure 1: Comprehensive Schematic of Hybrid Vulnerability Prioritization Architecture

3.4 Integration and Remediation Workflow Automation

To ensure that the hybrid prioritization framework delivers tangible value, it must be deeply integrated into the operational workflows of both the development and security operations teams. The framework is designed to automatically generate actionable remediation tickets within the organizational issue tracking system. When a vulnerability surpasses a predefined contextual risk threshold, the system automatically creates a defect ticket assigned to the appropriate development team. This ticket is enriched with comprehensive context, including the standardized vulnerability description, the exact file and line number identified by the static analysis, the network endpoint and payload utilized during the dynamic analysis, and a detailed explanation of the contextual risk score. Providing developers with this unified, corroborated data eliminates the need for them to manually piece together information from different security platforms, significantly accelerating the triage and remediation process. Furthermore, the framework continuously monitors the status of identified vulnerabilities. If a new deployment resolves the issue, the system automatically verifies the fix through targeted

dynamic re-testing and closes the associated ticket. This closed-loop automation ensures that the security posture is continuously assessed and that the development velocity is not hindered by administrative overhead.

4. Results and Discussion

4.1 Experimental Setup and Environmental Constraints

To evaluate the efficacy of the proposed hybrid prioritization framework, an extensive experimental validation was conducted within a simulated, enterprise-scale payment environment. The testbed was constructed using a prominent container orchestration platform, consisting of over two hundred distinct microservices designed to replicate the end-to-end flow of a modern financial transaction system. The application portfolio included user authentication gateways, transaction routing engines, fraud detection machine learning models, database access layers, and third-party integration proxies. The underlying codebase was a polyglot mixture of various modern programming languages, reflecting the heterogeneous nature of real-world enterprise architectures. Over a period of three months, standard development activities were simulated, including routine feature updates, dependency upgrades, and architectural modifications. Throughout this period, industry-standard static and dynamic application security testing tools were integrated into the continuous deployment pipeline. A baseline dataset was established by collecting all raw, unprioritized alerts generated by the standalone tools. Subsequently, the data was processed through the proposed hybrid correlation and contextual scoring framework. The objective of the experiment was to measure the reduction in total actionable alerts, assess the accuracy of the correlation engine, and evaluate the impact of the contextual scoring model on the prioritization of critical security flaws [19].

4.2 Performance Metrics and Comparative Analysis

The empirical analysis yielded substantial evidence supporting the effectiveness of the hybrid approach. During the experimental period, the standalone static analysis tools generated a massive volume of alerts, characterizing a significant portion of the codebase as potentially vulnerable. A detailed review indicated that a large majority of these alerts were either false positives or related to coding practices that, while suboptimal, did not present an exploitable security risk in the given deployment context. The standalone dynamic analysis tools generated fewer alerts, but frequently lacked the necessary detail for rapid remediation. When the raw datasets were processed through the hybrid framework, the correlation engine successfully linked a substantial percentage of the static and dynamic findings. Findings that lacked corroborating evidence from both domains were systematically deprioritized. The application of the contextual risk scoring mechanism further refined the dataset. Vulnerabilities located in highly exposed, mission-critical payment processing nodes were elevated to the top of the remediation queue, while those residing in isolated, backend administrative tools were downgraded. The comparative analysis demonstrated that relying solely on standardized baseline scoring resulted in a misallocation of remediation resources, as development teams would theoretically spend significant time patching theoretical vulnerabilities with negligible business impact. The hybrid framework, by contrast, provided a highly targeted list of actionable intelligence [20].

Table 2: Comparative Analysis of Vulnerability Prioritization Efficacy

Metric Category	Standalone Static Analysis	Standalone Dynamic Analysis	Hybrid Contextual Framework
Total Alerts Generated	Extremely High	Moderate	Low (Corroborated)
False Positive Rate	Substantial	Low	Negligible
Contextual Relevance	None	Minimal	Very High
Remediation Guidance	Specific (Code Level)	Vague (Network Level)	Comprehensive (Code and Network)
Prioritization Accuracy	Poor (Theoretical Risk)	Moderate (Exploitable Risk)	Excellent (Business Risk)

4.3 False Positive Reduction and Triage Efficiency

The most significant operational benefit observed during the experimental validation was the dramatic reduction in the false positive rate and the corresponding increase in triage efficiency. In a traditional security operations center, analysts must manually investigate hundreds of alerts daily, attempting to verify the exploitability and impact of each reported flaw. This manual process is heavily reliant on the individual expertise of the analyst and is fundamentally unscalable in environments with high deployment frequencies. The hybrid framework automates this verification process by utilizing the deterministic correlation rules. By presenting analysts and developers with a consolidated ticket containing both the theoretical flaw and the proof of its exploitability, the time required to understand and triage a vulnerability was reduced by an order of magnitude. Furthermore, the integration of business context into the scoring model fundamentally changed the dialogue between the security and development teams. In traditional paradigms, security teams often mandate the immediate remediation of all high-severity vulnerabilities, leading to friction with development teams focused on feature delivery. The hybrid framework provided a mathematically sound justification for prioritizing certain vulnerabilities over others, based on transparent factors such as network exposure and data sensitivity. This transparency fostered a more collaborative environment, as developers understood that the prioritized queue represented genuine, imminent threats to the payment infrastructure rather than a generic checklist generated by automated tooling. The reduction in alert noise and the improved clarity of the remediation directives directly contributed to a more resilient and secure deployment lifecycle.

5. Conclusion

The transition toward cloud-native microservices has revolutionized the operational capabilities of modern payment systems, but it has concurrently escalated the complexity of securing these distributed architectures. Standalone security testing methodologies, while foundational, are insufficient for managing the sheer volume and velocity of vulnerability data generated in continuous deployment environments. The pervasive issue of alert fatigue continuously undermines organizational security postures by obscuring critical threats beneath a deluge of false positives and decontextualized warnings. This research has presented a robust, automated framework for hybrid vulnerability prioritization that directly addresses these systemic challenges. By integrating static code analysis with dynamic runtime testing, the proposed architecture establishes a high-confidence correlation mechanism that validates theoretical vulnerabilities against exploitable attack surfaces. Crucially, the methodology advances beyond technical severity metrics by incorporating a dynamic contextual scoring algorithm that evaluates environmental exposure, data sensitivity, and the intrinsic business criticality of the affected microservices. The empirical validation demonstrated that this multi-dimensional approach significantly condenses the volume of actionable alerts, virtually eliminates false positives, and accelerates the remediation of genuine threats. Consequently, the framework enables security and development teams to allocate resources with precision, enhancing the overall security and compliance posture of massive-scale financial infrastructures.

6. Future Research Directions

While the proposed framework offers a significant advancement in vulnerability management for cloud-native payment services, several avenues for future research remain. The current correlation engine relies heavily on deterministic mapping and fuzzy logic based on standardized vulnerability classifications and endpoint topologies. Future iterations could explore the integration of advanced machine learning techniques, such as natural language processing, to automatically analyze the unstructured data within security reports and application logs, thereby uncovering deeper semantic relationships between disparate findings. Additionally, expanding the ingestion layer to include telemetry from interactive application security testing and runtime application self-protection tools could provide even greater granularity regarding application behavior and defensive mechanisms [21]. Further research is also required to adapt the contextual scoring models for serverless architectures, where the

abstraction of infrastructure introduces novel challenges in assessing environmental exposure and asset criticality. Ultimately, the continuous evolution of automated, intelligent prioritization frameworks will be essential for maintaining the integrity and trust of the global digital economy.

References

Hu, X., Wan, Z., & Zhou, Q. (2025). The Profitability Puzzle of On-Demand Carpooling: Pricing, Matching, and Route Compatibility. HKU Jockey Club Enterprise Sustainability Global Research Institute Paper, (2025/001).

Yang, Z., Ji, W., & Wang, Z. (2024). Adaptive joint configuration optimization for collaborative inference in edge-cloud systems. Science China Information Sciences, 67(4), 149103.

Zhao, H., Huang, H., & Li, X. (2025, November). PP-Pose: Privacy-Preserving Human Pose Estimation Using Random High-Frequency Channel Combinations. In Australasian Joint Conference on Artificial Intelligence (pp. 191-203). Singapore: Springer Nature Singapore.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. arXiv preprint arXiv:1606.06565.

Wang, S., Zhang, X., Wang, P., & Li, X. (2026). Design of Magnetic Sensor Array-Based PUFs for IoT Security. IEEE Transactions on Instrumentation and Measurement, 75, 1-9.

Zhou, D. (2025, December). M-VP2: Microservice-Oriented Vulnerability Patch Planning-A Cost-Aware Approachusing Multi-Agent Reinforcement Learning. In 2025 5th International Conference on Computer, Internet of Things and Control Engineering (CITCE) (pp. 248-254). IEEE.

Hu, X., Caldentey, R., & Vulcano, G. (2013). Revenue sharing in airline alliances. Management Science, 59(5), 1177-1195.

Li, X., Wang, P., Li, G., Ni, L., & Zhang, Y. (2023). Design of interface circuits and lightweight PUF for TMR sensors. IEEE Sensors Journal, 23(11), 11754-11761.

Zhou, D. (2026). AI-Driven Hybrid SAST–DAST–SCA–IAST Framework for Risk-Based Vulnerability Prioritization in Microservice Architectures.

Guo, Hanzhong, et al. "Real-time identity defenses against malicious personalization of diffusion models." arXiv preprint arXiv:2412.09844 (2024).

Chen, Y., Zhao, C., & Gao, Y. (2025). Blockchain applications in health insurance: a review of applications, challenges, and prospects. Frontiers in Blockchain, 8, 1699290.

Shen, B., Wang, Y., Shang, D., & Ning, T. (2026). Self-Commitment Latency: A Reward-Free Probe for Prompted Implicit Hacking. Publication date: 2026-06-04.

Wuyang Zhang and Shichao Pei. 2026. Your LLM Agent Can Leak Your Data: Data Exfiltration via Backdoored Tool Use. arXiv:2604.05432 [cs.CR] doi:10.48550/arXiv.2604.05432

Guo, J., Huang, S., Yao, Z., Zhang, Y., Lu, Y., Liu, J., ... & Wang, M. (2025). CryptoBench: A Dynamic Benchmark for Expert-Level Evaluation of LLM Agents in Cryptocurrency. arXiv preprint arXiv:2512.00417.

Yang, Z., Liang, B., & Ji, W. (2021). An intelligent end–edge–cloud architecture for visual IoT-assisted healthcare systems. IEEE internet of things journal, 8(23), 16779-16786.

Zhao, Y., & Han, Z. (2025, December). Safe and Adaptive Cloud Healing: Verifying LLM-Generated Recovery Plans with a Neural-Symbolic World Model. In 2025 7th International Academic Exchange Conference on Science and Technology Innovation (IAECST) (pp. 89-94). IEEE.

Yang, W., Qin, Y., & Yang, Y. (2021). Characterizing the multicast nature of malicious flows in CCN via SIS epidemic model. IEEE Systems Journal, 16(2), 2240-2250.

Li, S. (2025). Momentum, volume and investor sentiment study for us technology sector stocks—A hidden markov model based principal component analysis. PloS one, 20(9), e0331658.

Yang, Z., Ji, W., Guo, Q., Zhao, J., Zhao, C., Zhang, X., ... & You, Y. (2026). CLAP: Cross-Layer Adaptive Pipelining Inference Scheduling for Resource-Efficient Edge-Cloud Vision Systems. ACM Transactions on Architecture and Code Optimization.

Li, C., Chen, H., & Cai, W. (2026, April). From Slang to Standards: Consensus-Driven Airdrop Hunter Definition as a Baseline for Cryptocurrency Ecosystem Security and Governance. In Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (pp. 1-18).

Yang, Z., Ji, W., Guo, Q., & Wang, Z. (2023, October). Javp: Joint-aware video processing with edge-cloud collaboration for dnn inference. In Proceedings of the 31st ACM International Conference on Multimedia (pp. 9152-9160).