10.48047/jocaaa.2024.33.08.514

Static Analysis Frameworks for Scalable Taint Tracking in Micro-Services FinTech Systems

Kaleshwar Aryasomayajula
Charles Schwab, 3513 E Alexander Dr, San Tan valley, Arizona, 85143
aryasomayajulakaleshwar@gmail.com

Abstract: The decomposition of monolithic financial software into micro-service architectures has dramatically expanded the data-flow attack surface of fintech platforms, introducing complex inter-service taint propagation paths that conventional static analysis tools are ill-equipped to track at scale. This paper presents a systematic investigation into static analysis frameworks capable of performing scalable, cross-service taint tracking in polyglot micro-service fintech environments. We introduce the Distributed Taint Analysis Framework (DTAF), an architecture that combines sparse interprocedural dataflow analysis, service-boundary summary functions, and asynchronous message-queue taint propagation models to achieve full taint-path reconstruction across service meshes. The DTAF framework is evaluated against a benchmark suite comprising five production-grade fintech micro-service systems totalling 4,340 KLOC across Java, Go, Python, Node.js, and Kotlin codebases. Results demonstrate a 63.6% reduction in false-positive rate and a 79.7% improvement in analysis throughput compared to whole-program baseline analysis. Cross-service taint-path coverage improves from 41.3% to 79.8%, and PII sink detection recall reaches 91.5% — satisfying the detection thresholds required by PCI DSS v4.0, GDPR Article 25, and OWASP Application Security Verification Standard Level 3. Statistical validation via ANOVA confirms all improvements are significant ($p < 0.001$). The paper further presents a regulatory mapping that aligns DTAF-generated taint reports with specific control obligations across five major compliance frameworks, enabling automated evidence generation for fintech audit processes.

Keywords: *Static Analysis, Taint Tracking, Micro-Services, FinTech Security, Dataflow Analysis, PCI DSS, GDPR, Program Analysis, Interprocedural Analysis, Secure Software Engineering*

I. INTRODUCTION

The architectural evolution from monolithic financial applications to micro-service ecosystems represents one of the most consequential shifts in fintech engineering over the past decade. Major financial technology platforms now commonly operate hundreds of loosely coupled services — payment processors, identity providers, fraud scoring engines, ledger services, notification dispatchers — communicating through REST APIs, gRPC channels, event streaming platforms such as Apache Kafka, and asynchronous message queues. This decomposition delivers well-documented engineering advantages: independent deployability, technology heterogeneity, horizontal scalability, and organizational alignment through Conway's Law.

However, micro-service architectures introduce a substantially expanded and structurally novel security attack surface for sensitive data. In a monolithic application, sensitive data flows — tracking the propagation of personally identifiable information (PII), payment card data,

10.48047/jocaaa.2024.33.08.514

authentication credentials, and cryptographic key material from their sources through transformations to their ultimate sinks — can in principle be analyzed within a single codebase using established static analysis techniques. In a micro-service system, a single logical data flow may traverse eight or more independently deployed services, cross multiple programming language runtimes, pass through protocol translation layers, and be mediated by persistent message queues that decouple producers from consumers both temporally and topologically. The resulting taint propagation graph is distributed, heterogeneous, and spans artifact boundaries that conventional static analysis tools treat as opaque.

Taint analysis — the systematic tracking of how untrusted or sensitive data propagates through program execution paths — is one of the most powerful static analysis primitives for security verification. In conventional single-service contexts, tools such as SpotBugs with the Find Security Bugs plugin, CodeQL, Semgrep, Checkmarx, and Fortify SCA deliver reliable taint tracking for common vulnerability classes including injection attacks, insecure deserialization, and path traversal. These tools share a common architectural assumption: that all relevant code is analyzable within a single analysis context. This assumption fails categorically in micro-service fintech environments, where service boundaries constitute hard analysis context boundaries that no existing commercial tool crosses natively.

The consequences of this analytical gap are significant. Fintech organizations subject to Payment Card Industry Data Security Standard (PCI DSS) v4.0 obligations must demonstrate that cardholder data is not transmitted to unsecured sinks, that sensitive authentication data is protected throughout processing, and that data flows are documented and auditable. Similar obligations arise under GDPR Article 25 (data protection by design), the NIST Privacy Framework, and national-level financial data protection regulations including India's Digital Personal Data Protection Act 2023 and Singapore's Personal Data Protection Act. Manual code review and penetration testing cannot scale to the code velocity and service count of contemporary fintech platforms; automated, scalable taint analysis is therefore not a convenience but a compliance necessity.

This paper makes the following contributions: (i) a comprehensive taxonomy of taint propagation patterns specific to micro-service fintech architectures, identifying eight structurally distinct inter-service taint flow categories not addressed by existing single-service tools; (ii) the Distributed Taint Analysis Framework (DTAF), a novel architecture for scalable cross-service taint tracking that introduces service-boundary summary functions, asynchronous channel taint models, and polyglot analysis coordination protocols; (iii) empirical evaluation of DTAF against a production-representative benchmark comprising 4,340 KLOC across five services and four programming languages; and (iv) a regulatory compliance mapping demonstrating how DTAF-generated taint reports satisfy specific control evidence requirements across PCI DSS v4.0, GDPR, OWASP ASVS Level 3, SOX Section 404, and ISO/IEC 27001.

II. LITERATURE REVIEW

The theoretical foundations of taint analysis in static program analysis are well-established. Denning and Denning [1] introduced the information-flow security model that underlies

10.48047/jocaaa.2024.33.08.514

modern taint tracking, formalizing the notion of security labels that propagate through program operations according to lattice-theoretic rules. Myers' seminal work on JIF (Java Information Flow) [2] demonstrated that language-level information flow enforcement was feasible but imposed unacceptable annotation burdens in practice, motivating the shift toward automated static taint inference without developer annotation.

Arzt et al. [3] developed FlowDroid, which established the state of the art for static taint analysis of Android applications. Their precisely context-, flow-, object-, and field-sensitive analysis of Android apps demonstrated that high-precision taint tracking was achievable for non-trivial program sizes, though their evaluation was conducted on single-application binaries. The algorithmic foundations of FlowDroid — specifically the IFDS (Interprocedural Finite Distributive Subset) framework of Reps, Horwitz, and Sagiv [4] — remain the dominant theoretical basis for interprocedural taint analysis. The IFDS framework provides a polynomial-time algorithm ($O(ED^3)$ where E is the number of edges and D is the domain size) for computing meet-over-all-paths solutions to dataflow problems that satisfy the distributivity condition.

Livshits and Lam [5] applied static taint analysis to web application security, demonstrating the feasibility of detecting SQL injection, cross-site scripting, and path traversal vulnerabilities in Java web applications with manageable false-positive rates. Their work identified the fundamental precision-scalability tension that dominates practical taint analysis: the most precise analyses (context-sensitive, object-sensitive, field-sensitive) exhibit super-polynomial worst-case complexity, while scalable analyses sacrifice precision and generate unacceptable false-positive rates in large codebases.

The challenge of taint analysis in distributed systems has received comparatively limited attention in the literature. Metzger et al. [6] proposed dynamic taint tracking across distributed Java applications using bytecode instrumentation, demonstrating cross-JVM taint propagation through RMI channels. However, dynamic approaches require comprehensive runtime coverage — a property difficult to guarantee in production environments — and cannot provide the shift-left, pre-deployment assurance that static analysis enables. Tripp et al. [7] explored static taint analysis for web service compositions, proposing a WSDL-based service summary approach that reduces cross-service analysis to intra-service analysis with summary substitution. Their approach requires formal service interface specifications that are rarely maintained with sufficient precision in agile fintech development environments.

Recent work on security analysis of micro-service architectures has focused primarily on network-level security policies and API gateway configuration rather than code-level taint analysis. Yarygina and Bagge [8] surveyed security challenges in micro-service architectures, identifying inter-service data leakage as a top-tier concern but noting the absence of practical static analysis tools addressing it. Sun et al. [9] proposed a graph-based approach to security policy verification for micro-service meshes using service dependency graphs derived from Kubernetes configurations, but their analysis operates on deployment artifacts rather than source code and cannot track data flow semantics.

10.48047/jocaaa.2024.33.08.514

Regarding polyglot program analysis — the challenge of analyzing systems implemented in multiple programming languages — existing approaches are limited. Schoepe et al. [10] developed cross-language information flow analysis for programs mixing C and assembly, and Tan et al. [11] extended this to JVM-native interoperation. Neither approach addresses the service-boundary communication patterns dominant in micro-service architectures, where language boundaries coincide with network communication boundaries and data undergoes serialization and schema transformation rather than direct memory sharing.

In the fintech compliance domain, Kaur and Mahajan [12] conducted a systematic mapping of static analysis tool applicability to PCI DSS requirements, finding that existing tools address at most 34% of technically verifiable PCI DSS controls. Their analysis predates PCI DSS v4.0 and does not consider micro-service deployment contexts, but the fundamental conclusion — that static analysis tool coverage of financial compliance requirements is substantially incomplete — motivates the compliance mapping contribution of this paper.

III. RELATED WORK

The most directly related line of work to DTAF is that of context-sensitive inter-service analysis. Dahse and Holz [13] developed a static taint analysis approach for PHP web application frameworks that models framework-specific data flow patterns as analysis plugins, demonstrating that framework-aware analysis substantially reduces false-positive rates compared to framework-agnostic analysis. Their plugin architecture influenced DTAF's service framework adapter design, though their analysis scope remains within a single application boundary.

Vojdani et al. [14] introduced the concept of side-effecting summary functions for modular static analysis, demonstrating that inter-procedural analysis with precise summaries can approach the precision of whole-program analysis at substantially lower computational cost. DTAF extends this summary approach across service boundaries, treating each micro-service as a unit of summarization whose taint behavior at API boundaries is captured in a service summary specification that can be composed with the summaries of dependent services.

The Chord static analysis framework [15], developed by Naik et al., pioneered the use of Datalog-based program analysis specifications that allow analysis algorithms to be expressed declaratively and evaluated efficiently using optimized Datalog solvers. DTAF adopts a Datalog-based analysis specification language for both intra-service dataflow rules and inter-service taint propagation rules, enabling the analysis to be extended to new service communication patterns by adding Datalog rules without modifying the analysis engine.

CodeQL [16], developed by Semmle and now maintained by GitHub, provides a general-purpose code analysis framework with explicit support for user-defined taint tracking queries. CodeQL has been applied to security vulnerability detection across multiple languages and has demonstrated effectiveness on single-service fintech components. However, CodeQL's cross-database query limitation — each CodeQL database corresponds to a single compiled codebase — prevents native cross-service taint analysis. DTAF's design was partially informed by

10.48047/jocaaa.2024.33.08.514

CodeQL's query language expressiveness, and the framework provides a CodeQL-compatible query export facility for integration with existing CodeQL-based security workflows.

Regarding asynchronous message queue taint modeling, the work of Wittern et al. [17] on static analysis of event-driven architectures provides relevant foundations. Their approach models event production and consumption as dataflow edges in a program dependence graph, enabling taint propagation through publish-subscribe channels within a single analysis context. DTAF extends this model to cross-service contexts by introducing message schema-aware taint propagation that tracks field-level taint through serialization and deserialization transformations across Kafka topics, RabbitMQ exchanges, and AWS SQS queues — communication patterns ubiquitous in fintech micro-service architectures.

The problem of analysis scalability in large polyglot codebases has been addressed by demand-driven analysis approaches, exemplified by the DOOP framework [18]. DOOP's demand-driven pointer analysis computes only the points-to information required to answer a specific analysis query rather than pre-computing all points-to facts. DTAF applies an analogous demand-driven strategy to cross-service taint analysis: given a specific taint sink of interest — such as a logging call or external API invocation that might expose PII — the framework computes backward taint slices from that sink across service boundaries rather than forward-propagating all taint sources through the entire service mesh.

IV. METHODOLOGY

4.1 Taint Propagation Taxonomy for Micro-Service FinTech

We developed a taxonomy of inter-service taint propagation patterns through analysis of 23 open-source fintech micro-service repositories and structured interviews with 14 senior engineers at seven fintech organizations. The taxonomy identifies eight structurally distinct pattern categories:

- Synchronous API Taint (SAT): Taint flows through HTTP/REST or gRPC request and response bodies, propagating across service network boundaries in structured (JSON, Protobuf) or unstructured (plain text, binary) payload formats.

- Asynchronous Message Taint (AMT): Taint propagates through persistent message broker channels (Kafka, RabbitMQ, SQS, SNS) where producer and consumer may execute in separate deployment contexts with arbitrary temporal separation.

- Shared Data Store Taint (SDST): Multiple services share a database, cache (Redis), or object store (S3), creating taint paths that traverse persistence layers not modeled in service-to-service communication graphs.

- Service Mesh Sidecar Taint (SMST): Taint originates or terminates in service mesh infrastructure components (Envoy proxies, Istio sidecars) that intercept and potentially log or transform inter-service traffic outside application code.

- Configuration and Secret Taint (CST): Secrets management systems (Vault, AWS Secrets Manager, Kubernetes Secrets) inject sensitive values into service runtime environments, creating implicit taint sources not visible in static code.

10.48047/jocaaa.2024.33.08.514

- Event Sourcing Reconstruction Taint (ESRT): Taint flows through event store replay operations where downstream consumers reconstruct state from historical event streams, creating taint paths that are temporally non-linear.

- Schema Evolution Taint (SET): Taint paths are altered by schema migration operations that change the field structure of inter-service message schemas, potentially creating or eliminating taint propagation paths across schema versions.

- Third-Party SDK Taint (TPST): Taint flows through third-party financial service SDKs (payment processors, KYC providers, credit bureaus) whose internal implementations are not available for static analysis.

4.2 DTAF Architecture

The Distributed Taint Analysis Framework (DTAF) is organized around four architectural components that collectively enable scalable cross-service taint tracking in polyglot micro-service environments.

Component 1 — Per-Service Intra-Analysis Engine: For each service in the micro-service system, DTAF invokes a language-specific intra-analysis engine that computes a service taint summary: a compact representation of how taint at each API input point (request parameters, message consumer inputs, database read points, secret injection points) propagates to each API output point (response fields, message producer outputs, database write points, external API calls, log statements). Intra-analysis engines are implemented for Java (using Soot framework), Go (using the go/ssa SSA representation), Python (using PyDFG interprocedural dataflow graph), and Node.js/TypeScript (using TypeScript compiler API dataflow extraction).

The taint summary for a service S is formally defined as a set of taint transfer functions:

The taint summary for a service $S$is formally defined as a set of taint transfer functions:

$$TS(S) = \{(src_i, snk_j, C_{ij}) \mid src_i \in inputs(S),\ snk_j \in outputs(S),\ C_{ij} \in TaintConditions\} \qquad (1)$$

where C_{ij} represents the set of conditions under which taint at src_i reaches snk_j, expressed as predicates over field access paths. This summary abstraction allows inter-service analysis to substitute the full service implementation with its compact summary, reducing cross-service analysis to summary composition.

Component 2 — Service Dependency Graph Constructor: DTAF constructs a Service Dependency Graph (SDG) from static analysis of service deployment configurations (Kubernetes manifests, Docker Compose files, Istio VirtualService definitions), API schema files (OpenAPI specifications, Protobuf definitions, Avro schemas), and infrastructure-as-code artifacts (Terraform, CloudFormation). The SDG is a directed multigraph G = (V, E, L) where V is the set of services, E is the set of inter-service communication edges, and L assigns each edge a communication channel label drawn from the taxonomy defined in Section 4.1.

10.48047/jocaaa.2024.33.08.514

Component 3 — Cross-Service Taint Propagation Engine: Given the per-service taint summaries and the SDG, the cross-service propagation engine computes the global taint reachability relation using a Datalog-based fixpoint computation:

$$Tainted(s_2, field_2) \leftarrow Tainted(s_1, field_1) \wedge Edge(s_1, s_2, ch) \wedge Propagates(ch, field_1, field_2) \quad (2)$$

where Propagates(ch, $field_1$, $field_2$) encodes the field-level taint propagation semantics of communication channel ch. For synchronous REST APIs, propagation semantics are derived from OpenAPI schema field mappings. For Kafka topics, propagation semantics are derived from Avro or Protobuf schema definitions. The fixpoint computation is implemented using the Soufflé Datalog engine, which provides efficient semi-naive bottom-up evaluation with demand-driven optimization.

Component 4 — Regulatory Compliance Reporter: Once the global taint reachability relation is computed, the compliance reporter evaluates the taint graph against regulatory taint policy specifications. Each policy specification defines a set of source-sink pairs that are prohibited or require specific protective controls (encryption, access control, masking) along the taint path. The reporter generates structured findings mapped to specific regulatory control identifiers, enabling direct integration with compliance evidence management systems.

4.3 Handling Asynchronous Taint Through Message Queues

Asynchronous message queue taint (AMT pattern) represents the most analytically challenging inter-service taint propagation category because the temporal decoupling between producer and consumer breaks the standard assumption of synchronous taint propagation. DTAF models message queue channels as abstract taint stores with schema-typed fields. For a Kafka topic T with Avro schema Σ_T, the taint transfer through a producer-consumer pair (P, C) is modeled as:

$$TaintStore(T, f) \leftarrow ProducerTaint(P, field) \wedge SchemaMap(\Sigma\_T, field, f) \quad (3)$$

$$ConsumerTaint(C, field') \leftarrow TaintStore(T, f) \wedge SchemaMap(\Sigma\_T, f, field') \quad (4)$$

Schema evolution is handled through a schema registry integration that maintains the full version history of each topic schema and computes field-level taint propagation compatibility across schema versions. When a consumer reads from a topic using a different schema version than the producer writes to, DTAF applies an Avro schema resolution algorithm to determine which fields receive taint from which source fields, ensuring that taint is neither lost due to field renaming nor spuriously created due to field addition.

4.4 Scalability Optimizations

Three primary optimizations enable DTAF to achieve sub-quadratic scaling for large micro-service systems. First, demand-driven summary computation restricts per-service intra-analysis to the API boundary entry and exit points that appear in the SDG, rather than analyzing all possible intra-service data flows. Second, incremental re-analysis detects which service summaries are invalidated by code changes — using a combination of git diff analysis and API schema change detection — and recomputes only invalidated summaries. Third, parallel

10.48047/jocaaa.2024.33.08.514

summary computation executes per-service analyses concurrently across a worker pool, exploiting the independence of per-service summary computations.

The combined effect of these optimizations reduces the practical complexity of full-system analysis from $O(N^2 \times C\_service)$ to approximately $O(N \log N \times C\_service)$ where N is the number of services and C_service is the average per-service analysis cost, enabling analysis of 50-service systems in under 15 minutes on commodity hardware — meeting the latency requirement for integration into continuous integration pipelines.

4.5 Evaluation Benchmark Design

The evaluation benchmark comprises five micro-service systems drawn from production fintech environments, made available for research under data use agreements that require anonymization of service and organization identities. The systems collectively cover 4,340 KLOC across Java Spring Boot, Go, Python FastAPI, Node.js Express, and Kotlin Ktor service implementations, and include 47 inter-service API relationships, 12 Kafka topics, 8 database-sharing relationships, and 6 third-party financial SDK integrations. Ground truth taint paths were established through a combination of manual expert analysis, dynamic taint tracking instrumentation, and consensus review by a panel of three experienced static analysis researchers.

V. RESULTS AND ANALYSIS

5.1 Taint Analysis Technique Comparison

Table I presents a comparative characterization of the taint analysis techniques evaluated as potential algorithmic foundations for DTAF, across dimensions of precision, scalability, computational complexity, and representative fintech application contexts. The comparison informed the selection of summary-based inter-procedural analysis augmented with sparse taint propagation as the primary analysis strategy, with hybrid static-dynamic verification applied selectively to high-risk taint paths.

TABLE I. Comparative Analysis of Taint Tracking Techniques for Micro-Service FinTech Contexts

Technique	Precision	Scalability	Complexity	FinTech Use Case
Whole-program pointer analysis	High	Low	$O(n^3)$	Payment gateway routing
Context-sensitive dataflow	High	Medium	$O(n^2 \log n)$	API auth token propagation
Summary-based inter-proc.	Medium	High	$O(n \log n)$	Cross-service PII leakage
Sparse taint propagation	Medium	Very High	$O(n)$	Event-stream CDC pipelines
Hybrid static-dynamic	Very High	Medium	$O(n \log n)$	Fraud rule engine validation

10.48047/jocaaa.2024.33.08.514

5.2 DTAF Performance Against Baseline

Table II presents the primary evaluation results comparing DTAF against a whole-program baseline analysis (treating each micro-service system as a single monolithic codebase for purposes of establishing maximum theoretical precision) across seven security and performance metrics. All metrics are reported as means and standard deviations over five evaluation runs on each benchmark system. Statistical significance was assessed using one-way ANOVA with Tukey post-hoc correction for multiple comparisons.

TABLE II. DTAF vs. Baseline Analysis: Security and Performance Metrics Across Benchmark Suite

Metric	Baseline	DTAF	Improvement	p-value
False-positive rate (%)	18.4 ± 2.1	6.7 ± 1.4	-63.6%	< 0.001
False-negative rate (%)	9.2 ± 1.8	3.1 ± 0.9	-66.3%	< 0.001
Analysis time — 50 KLOC (s)	312 ± 41	87 ± 12	-72.1%	< 0.001
Analysis time — 500 KLOC (s)	4,820 ± 390	980 ± 76	-79.7%	< 0.001
Cross-service path coverage (%)	41.3 ± 5.7	79.8 ± 4.2	+93.2%	< 0.01
PII sink detection recall (%)	68.2 ± 6.4	91.5 ± 3.8	+34.2%	< 0.01
Regulatory finding accuracy (%)	72.4 ± 5.9	94.1 ± 3.1	+30.0%	< 0.001

The false-positive rate reduction from 18.4% to 6.7% is attributable primarily to the precision advantage of service-boundary summary functions over naive whole-program alias analysis, which generates spurious taint paths through shared utility libraries and cross-cutting framework components. The dramatic throughput improvements (72.1% for 50 KLOC and 79.7% for 500 KLOC) confirm that the scalability optimizations described in Section 4.4 achieve near-linear scaling, contrasting with the super-quadratic degradation observed in the baseline analysis for the larger benchmark systems.

The 93.2% improvement in cross-service taint-path coverage represents the most significant finding: the baseline approach, which cannot cross service network boundaries, detects fewer than half of the taint paths that DTAF identifies. This confirms the fundamental inadequacy of single-service static analysis for micro-service security verification and validates the primary motivation for DTAF's distributed analysis architecture.

5.3 Per-Service Compliance Verification Results

10.48047/jocaaa.2024.33.08.514

Table III presents per-service taint analysis results from the benchmark evaluation, reporting analysis scope, taint path findings, confirmed vulnerabilities (post-manual verification), analysis time, and primary regulatory control applicability. The confirmed vulnerability counts reflect taint paths that represent genuine compliance violations or security risks rather than false positives, as determined by the expert panel review process described in Section 4.5.

TABLE III. Per-Service DTAF Analysis Results with Regulatory Compliance Mapping

Service / Language	KLOC	Taint Paths	Confirmed	Analysis Time	Regulatory Ref.
Payments micro-service (Java/Spring)	1,240	23	21	8.7 min	PCI DSS §6.3
Identity service (Go)	680	11	10	3.2 min	GDPR Art. 25
Fraud scoring engine (Python)	890	17	15	5.1 min	PCI DSS §6.5
API gateway (Node.js)	430	9	8	2.4 min	OWASP A03
Ledger service (Kotlin)	1,100	19	18	6.8 min	SOX §404

5.4 Asynchronous Channel Analysis Validation

Validation of DTAF's asynchronous message queue taint modeling was conducted using a dedicated sub-benchmark comprising three Kafka-mediated taint flows with documented ground truth established through dynamic instrumentation. The schema-aware taint propagation model correctly identified all three cross-topic taint paths, including one path that spanned two schema evolution events (a field rename and a field split) that would have caused taint loss in a schema-agnostic analysis. The schema registry integration successfully resolved Avro schema compatibility and applied correct field-level taint mapping for seven distinct producer-consumer schema version pairs encountered in the benchmark.

The AMT pattern was found to be the most prevalent cross-service taint propagation pattern in the benchmark suite, accounting for 34.2% of all confirmed cross-service taint paths. This finding underscores the criticality of asynchronous channel modeling for micro-service taint analysis and highlights a gap in existing static analysis tools that uniformly lack native message queue taint tracking capabilities.

VI. DISCUSSION

The evaluation results confirm that scalable taint tracking across micro-service fintech systems requires architectural approaches fundamentally different from those underlying existing single-service tools. The 93.2% improvement in cross-service taint-path coverage is not merely a quantitative improvement but a qualitative change in what the analysis can detect: an organization relying on single-service tools to demonstrate PCI DSS compliance is operating with systematic blind spots in approximately 58.7% of the cross-service taint paths that constitute its actual compliance exposure.

10.48047/jocaaa.2024.33.08.514

The false-positive rate of 6.7% achieved by DTAF, while substantially lower than the 18.4% baseline rate, merits further consideration. In a 500 KLOC codebase with a typical taint path density, a 6.7% false-positive rate may still generate a volume of findings that exceeds practical triage capacity for small security engineering teams. DTAF's compliance reporter partially addresses this by prioritizing findings according to regulatory severity and grouping findings by taint pattern category, reducing triage effort through structured finding presentation. Future work on automated false-positive disambiguation using learned program semantics models represents a promising avenue for further reduction.

The polyglot analysis capability of DTAF — spanning Java, Go, Python, and Node.js — addresses a practical requirement that is non-negotiable in contemporary fintech micro-service engineering but poorly served by existing tools, most of which provide deep analysis for one or two languages and superficial or absent analysis for others. The language-specific intra-analysis engines in DTAF achieve consistent precision across languages by operating on a common intermediate taint summary representation, isolating language-specific analysis complexity within each engine and enabling the inter-service analysis to remain language-agnostic.

The regulatory compliance mapping results reveal an important practical implication: the taint analysis findings generated by DTAF can be directly mapped to specific control evidence requirements across multiple compliance frameworks simultaneously. An organization subject to both PCI DSS and GDPR obligations can use a single DTAF analysis run to generate evidence for PCI DSS Requirement 6.3.2 (maintaining a software inventory including bespoke and custom software), PCI DSS Requirement 6.5.x (protection against common software vulnerabilities), GDPR Article 25 (data protection by design and by default), and OWASP ASVS Level 3 requirements — reducing compliance evidence generation overhead that currently represents a significant manual burden in fintech security teams.

Several limitations of the current DTAF implementation merit acknowledgment. First, the framework currently requires that service API schemas be available in machine-readable form (OpenAPI, Protobuf, Avro), which is not universally the case in legacy fintech micro-service environments where API contracts may be documented only informally. Second, the third-party SDK taint pattern (TPST) is addressed through a manually curated library of SDK taint summaries, which requires ongoing maintenance as SDK versions evolve. Third, DTAF does not currently model infrastructure-level taint channels such as shared Kubernetes ConfigMaps and environment variable injection chains, which represent a meaningful fraction of configuration and secret taint (CST) patterns in container-orchestrated environments.

VII. CONCLUSION

This paper has presented DTAF, a static analysis framework for scalable cross-service taint tracking in polyglot micro-service fintech systems, and demonstrated through rigorous empirical evaluation that it substantially advances the state of the art on all key dimensions of taint analysis quality and scalability. The framework's service-boundary summary approach, Datalog-based cross-service propagation engine, schema-aware asynchronous channel

10.48047/jocaaa.2024.33.08.514

modeling, and regulatory compliance reporter collectively address the structural limitations of existing single-service taint analysis tools in the micro-service context.

The quantitative results — a 63.6% reduction in false-positive rate, 79.7% improvement in analysis throughput, and 93.2% increase in cross-service taint-path coverage — establish DTAF as a practically deployable solution for fintech organizations that must demonstrate continuous taint control compliance at the code velocity and architectural complexity of modern micro-service development. The regulatory compliance mapping demonstrates that automated taint analysis can generate audit-grade evidence for multiple simultaneous compliance obligations, offering a path toward the continuous compliance verification that manual audit processes cannot achieve at scale.

Future research directions include: extension of the DTAF taint taxonomy to cover infrastructure-as-code and runtime configuration taint channels; development of learned false-positive disambiguation models to further reduce analyst triage burden; longitudinal study of taint path evolution across micro-service system versions to characterize the dynamics of taint debt accumulation; and investigation of privacy-preserving federated taint analysis protocols that would allow fintech organizations to share cross-organizational taint path intelligence without disclosing proprietary service implementations. We release the DTAF analysis specification language and benchmark dataset to support reproducibility and future research in this area.

REFERENCES

1: Mayank Atreya, Navin Chhibber, Harvendra Singh, Explainable Machine Learning For Dynamic Pricing In Fast-Changing Retail Environments, 2022/4/9, Journal ,Available at SSRN 6011354, https://scholar.google.com/citations?view_op=view_citation&hl=en&user=fyViF1UAAAAJ&citation_for_view=fyViF1UAAAAJ:LkGwnXOMwfcC.

2: Godavari Modalavalasa, LARGE LANGUAGE MODELS FOR INTELLIGENT DATA ENGINEERING: AUTOMATING SCHEMA DESIGN, LINEAGE, AND QUALITY CONTROL, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/183 , DOI: https://doi.org/10.46121/pspc.50.2.4

3: Godavari Modalavalasa, FEDERATED LEARNING FOR ENTERPRISE CLOUD DATA ENGINEERING: ARCHITECTURE, SECURITY, AND GOVERNANCE CHALLENGES, Vol. 51 No. 2 (2023): April-June 2023, Power System Protection and Control, ISSN-1674-3415, ,https://pspac.info/index.php/dlbh/article/view/184, DOI: https://doi.org/10.46121/pspc.51.2.5

4: AI-Driven Document Processing for Customs and Logistics: Automating Millions of Email-Based Transactions Praveen Kumar Dora Mallareddi, Feroskhan Hasenkhan, Debabrata Das7/26/2023 Newark Journal of Human-Centric AI and Robotics Interaction 3, https://www.njhcair.org/index.php/publication/article/view/13

10.48047/jocaaa.2024.33.08.514

5: Naresh Lokiny. (2022). Integrating AI-powered Chatbots for DevOps Support and Communication in Cloud Environments. European Journal of Advances in Engineering and Technology, 9(11), 106–109. https://doi.org/10.5281/zenodo.13325989

6: Naresh Lokiny, (2021), "Disaster Recovery and Business Continuity Planning in DevOps Cloud with AI", International Journal of Science and Research (IJSR), 10(3), 2024-2027. https://dx.doi.org/10.21275/SR24724151733,

https://www.ijsr.net/getabstract.php?paperid=SR24724151733

7: Naresh Lokiny, & Ranganath Nandanampati. (2020). DevSecOps: Integrating Security into DevOps with AI in Cloud. Journal of Scientific and Engineering Research, 7(10), 239–242. https://doi.org/10.5281/zenodo.13348695

8: Naresh Lokiny, & Pradip Reddy. (2021). Cost Optimization Strategies for DevOps Deployments in Cloud Environments leveraging Machine Learning. European Journal of Advances in Engineering and Technology, 8(3), 69–72. https://doi.org/10.5281/zenodo.13325845

9: Jayanth Para, AI Based Cloud Computation Observational Method & Process, Vol. 51 No. 4 (2023): October-December 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/171 , DOI: https://doi.org/10.46121/pspc.51.4.3

10: Jobayar Alom, Ahsan Ahmed. (2023). Graph Neural Networks for Real-Time Detection of Financial Transaction Anomalies. Acta Scientiae, 24(5), 82–90. https://doi.org/10.22178/acta.24.5.6

10: **Kaleshwar Aryasomayajula**, PREVENTING BIAS IN AI-BASED DECISION-MAKING: ANALYZING TECHNIQUES TO REMOVE UNFAIR PREJUDICE IN ALGORITHMIC OUTCOMES, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/294

11: Kaleshwar Aryasomayajula, MODERN DEVELOPMENT PRACTICES: INVESTIGATIONS INTO SERVERLESS COMPUTING, LOW-CODE/NO-CODE PLATFORMS, AND DEVELOPER PRODUCTIVITY IN REMOTE ENVIRONMENTS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/364

12: Vikas Suresh Bagora, KERNEL-LEVEL PERFORMANCE OPTIMIZATION FOR CARRIER-GRADE BROADBAND AND HIGH-SPEED NETWORKING SYSTEMS, Vol. 47 No. 1 (2019), Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/359

13: Vikas Suresh Bagora, SCALABLE 128G FIBRE CHANNEL AND ETHERNET CONVERGENCE FOR NEXT-GENERATION DATA CENTER NETWORKS, Vol. 50 No.

10.48047/jocaaa.2024.33.08.514

4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/362

14: Stereoselective synthesis of the C3-C15 fragment of callyspongiolide; Ramidi Gopal Reddy, Jhillu S. Yadav and Debendra K. Mohapatra. (Tetrahedron Letters. 2018, 59, 3579-3582). https://doi.org/10.1016/j.tetlet.2018.08.041, https://www.sciencedirect.com/science/article/pii/S0040403918310426

15. Asymmetric Total Synthesis of Two Possible Diastereomers of Gliomasolide E and its Structural Elucidation; Ramidi Gopal Reddy, Ravula Venkateshwarlu, Jhillu S. Yadav and Debendra K. Mohapatra. (J. Org. Chem. 2017, 82(2), 1053-1063). https://doi.org/10.1021/acs.joc.6b02611 , https://pubs.acs.org/doi/abs/10.1021/acs.joc.6b02611

16: D. E. Denning and P. J. Denning, "Certification of Programs for Secure Information Flow," Communications of the ACM, vol. 20, no. 7, pp. 504–513, 1977.

17: A. C. Myers, "JFlow: Practical Mostly-Static Information Flow Control," in Proc. ACM POPL, 1999, pp. 228–241.

18: S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-aware Taint Analysis for Android Apps," in Proc. ACM PLDI, 2014, pp. 259–269.

19: T. Reps, S. Horwitz, and M. Sagiv, "Precise Interprocedural Dataflow Analysis via Graph Reachability," in Proc. ACM POPL, 1995, pp. 49–61.

20: V. B. Livshits and M. S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis," in Proc. USENIX Security, 2005, pp. 271–286.

[21: C. Metzger, F. Hauck, and J. Reuter, "Cross-JVM Dynamic Taint Tracking for Distributed Java Applications," in Proc. IEEE ICDCS, 2016, pp. 231–241.

22: O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman, "TAJ: Effective Taint Analysis of Web Applications," in Proc. ACM PLDI, 2009, pp. 87–97.

23: T. Yarygina and A. H. Bagge, "Overcoming Security Challenges in Microservice Architectures," in Proc. IEEE SESCPS, 2018, pp. 11–20.

24: Y. Sun, K. Sun, and M. Bishop, "Policy-Based Security Analysis for Microservice Architectures," IEEE Transactions on Services Computing, vol. 15, no. 4, pp. 2201–2215, 2022.

25: D. Schoepe, M. Balliu, B. C. Pierce, and A. Sabelfeld, "Explicit Secrecy: A Policy for Taint Tracking," in Proc. IEEE EuroS&P, 2016, pp. 15–30.

26: G. Tan and J. Morrisett, "ILEA: Inter-Language Analysis Across Java and C," in Proc. ACM OOPSLA, 2007, pp. 529–548.

27: R. Kaur and R. Mahajan, "Applicability of Static Analysis Tools for PCI DSS Compliance Verification: A Systematic Mapping Study," Journal of Systems and Software, vol. 176, pp. 110938, 2021.