Ingénierie des Systèmes d'Information
Vol. 31, No. 10, October, 2026, pp. 3427-3440
Journal homepage: http://iieta.org/journals/isi

A Graph-Theoretic Framework for Detecting Race Conditions in FHIR-Based Healthcare Interoperability

Jawad Mohammed*, Gahangir Hossain

Department of Data Science, University of North Texas, Denton 76203, United States

Corresponding Author Email: JawadMohammed@my.unt.edu

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/isi.311016

Received: 3 April 2026
Revised: 17 August 2026
Accepted: 9 September 2026
Available online: 31 October 2026

Keywords:
FHIR, healthcare interoperability, race-condition detection, graph-theoretic modeling, concurrent access, TOCTOU vulnerability

ABSTRACT

Concurrent access to shared resources in Fast Healthcare Interoperability Resources (FHIR)-based systems can produce race conditions that authentication controls alone do not address. This study introduces the FHIR Resource Access Graph (FRAG), a formal graph-theoretic model for representing resource-access events and detecting three classes of concurrency defects: Simultaneous Write Conflict (SWC), time-of-check-to-time-of-use (TOCTOU), Authorization Violation (TAV), and Cascading Update Race (CUR). The model incorporates a synchronization constraint set and a concurrency window, and establishes detector soundness and conditional completeness when the synchronization constraints are correctly specified. A three-pass graph-traversal detector was implemented and evaluated using 1,500 synthetic FHIR R4 transaction logs under sequential, fully concurrent, and partially synchronized conditions. Under full concurrency, FRAG achieved an overall F1-score of 90.0%, compared with 25.5% for a time-window baseline; under partial synchronization, the corresponding scores were 79.0% and 25.4%. No false positives were observed in the sequential control condition. Additional experiments showed that processing time increased from 1.7 ms for 1,000 events to 0.51 s for 250,000 events under bounded per-resource contention. Detection performance was also sensitive to timestamp granularity, with F1 declining substantially at 100 ms resolution. These results demonstrate the feasibility of graph-based race-condition detection under controlled synthetic conditions. However, the findings do not establish performance in operational clinical environments, and validation using real FHIR access logs remains necessary.

1. INTRODUCTION

The problem addressed in this paper did not come with a label. It evolved from recognizing that two research communities, one in healthcare informatics and the other in systems security, have dealt with related problems for decades without formally connecting them. One community studies the flow of patient information across heterogeneous healthcare systems; the other studies what happens when multiple processes contend for shared memory without proper synchronization. The combination of these two problems has, to our knowledge, not been formally addressed before.

Consider the following scenario, representative of any Fast Healthcare Interoperability Resources (FHIR)-based interoperability deployment: a patient presents to an emergency room. A laboratory system has just completed a drug-sensitivity test and is updating an AllergyIntolerance resource on the FHIR server. At the same time, an electronic health record system is reading that same resource to check a new prescription order. At the same time, a clinical decision-support system is reading the patient's allergy history to generate a contraindication alert. Three systems, one resource, no lock, no mutex, no sequencing; the FHIR specification does not define one [1].

The problem of the race condition is sixty years old. Dijkstra formalized the critical-section problem in 1965, and the three conditions for correct concurrent access—mutual exclusion, progress, and bounded waiting—have been foundational to operating-systems design ever since [2]. What is notable about the healthcare case is not the novelty of the underlying problem but the fact that a large data-exchange infrastructure has been built on restful application programming interface (API), a technology that offers none of the synchronization primitives operating system (OS) research has spent sixty years developing. An OS process accessing shared memory can acquire a mutex, whereas a FHIR client accessing a shared patient resource has access only to an optional ETag header, which is not enforced by most implementations [3].

The result is a vulnerability class that is neither a network attack nor an authentication failure, but a timing vulnerability, a property of how multiple independently correct systems interact without coordination. A patient can receive a contraindicated drug not because any individual system failed, but because three systems each acted correctly in a way their individual correctness could not prevent.

The specific, bounded claim of this paper is: race-condition vulnerabilities in FHIR-based healthcare interoperability systems are formally characterizable as properties of a directed access graph, and this characterization yields a practical detection framework. We do not claim to have solved FHIR's

synchronization problem; we claim to have formally stated its detection problem, and shown that statement to be both precise and tractable and demonstrated its computational scalability and operational deployability.

1.1 Contributions

- A structured literature survey, backed by a documented search protocol (Section 2.4), that formally identifies the gap between OS concurrency research and healthcare interoperability security research.
- The FHIR Resource Access Graph (FRAG), a formally defined directed graph with an explicit synchronization constraint set S and an explicit, non-adversarial system and threat model (Section 3.5).
- Three formally defined, clinically relevant race-condition classes (Simultaneous Write Conflict (SWC), Authorization Violation (TAV), and Cascading Update Race (CUR)), with structural correspondence lemmas and new soundness and δ-completeness results, the latter valid under an accurately specified synchronization set S (Section 4.2).
- A simulation-based evaluation on 1,500 synthetic FHIR R4 transaction logs across three concurrency conditions with a sensitivity analysis (Section 9.2), a stronger baseline comparison (Section 5.6), a deployment architecture (Section 8), and a theoretical and empirical computational-complexity analysis (Section 9), with open-sourced code.

2. RELATED WORK

The relevant literature falls into two categories that never reference each other. The bridge between the two is what this paper aims to build.

2.1 Concurrency detection in systems software

This subsection surveys concurrency-detection research from systems software as background precedent for a graph-theoretic detection paradigm. None of the studies below target FHIR, HL7, or any healthcare interoperability platform; each is included only to establish what a structural or learning-based race-condition detector looks like in a domain with a mature history of concurrency-bug patches, not because the underlying systems are methodologically close to FHIR's representational state transfer (REST)-based concurrency model. Section 2.2 then reviews the FHIR- and healthcare-specific literature directly. Upadhyay et al. [4] give a taxonomy of race-condition detectors, static, dynamic, and hybrid spanning 1993–2023. Their finding that data-race bugs make up roughly 80% of concurrency bugs frames the scale of the problem this paper brings into a new domain.

Li et al. [5] introduce LR-Miner, a field-graph-based static analyzer that mines locking rules from the Linux kernel source across 17.2 million lines of code. LR-Miner is relevant in a negative sense: its entire premise is inferring rules from existing code and patch patterns, and there is no equivalent history of locking conventions or race-condition patches in healthcare middleware to mine.

ConSynergy [6] combines static analysis with large language model (LLM)-based semantic reasoning in a four-stage pipeline, reporting an F1 of 0.867 on general benchmarks. It is the most similar prior work in spirit, using structured representations of concurrent access, but it operates on compiled source code, requires a training corpus of known concurrency bugs, and has not been validated in any healthcare context, none of which FRAG requires.

KNighter [7] constructs static analyzers from bug-fix patches using LLMs. Like LR-Miner, it is patch-dependent and cannot detect race conditions in systems where no incidents have been documented yet, which describes the current state of healthcare middleware.

Gong et al. [8] propose Snowcat, a coverage-guided kernel concurrency tester using a learned predictor, evaluated exclusively on OS kernel workloads rather than any healthcare or REST API context. Their finding that kernel concurrency bugs require domain-specific concurrency models motivated the present work: FHIR's concurrency model, asynchronous RESTful HTTP with optional ETags, is quite different from kernel threading and requires its own domain-specific analysis. Collectively, studies [4-8] establish that structural and learning-based concurrency detectors are a mature paradigm in systems software, but one with no existing instantiation for FHIR or HL7-based interoperability; that gap is the subject of Sections 2.2–2.3.

2.2 Healthcare interoperability security

TXOne Networks [3] document authentication flaws in FHIR systems, including the XML external entity (XXE) injection vulnerability Common Vulnerabilities and Exposures (CVE)-2024-52007 [9]. This is the most technically detailed recent analysis of FHIR security, but it assumes sequential data access throughout; the term "concurrent access" does not appear. The implicit assumption that only one system reads or writes at a time is architecturally false in any real multi-system deployment.

The distributed ledger technology (DLT)-based electronic health record (EHR) architecture [10] addresses data integrity through blockchain immutability, a genuine contribution, but is limited to the storage layer. Race conditions occur at the access layer, between the time a process reads a resource and the time it acts on the result, which a blockchain storage layer does not address.

A broader body of recent, peer-reviewed literature addresses FHIR interoperability and healthcare data security without treating concurrent access as a correctness property. On the interoperability side, Maguluri et al. [11] address cybersecurity risks associated with Bring Your Own Device (BYOD) practices in Healthcare 5.0, while Velagala and Hossain [12] explore early detection of insider cyber threats and human factors in Healthcare 5.0. Both studies focus on healthcare cybersecurity risks rather than race conditions arising from concurrent reads and writes of FHIR resources. Two systematic reviews of the FHIR literature, broadly Ayaz et al.'s [13] review of FHIR implementations, applications, and challenges and Tabari et al.'s [14] more recent scoping review of FHIR-based data models, synthesize dozens of FHIR deployment studies without either identifying concurrency or race conditions as a correctness dimension; Bae et al.'s [15] blockchain-based FHIR health-information-exchange platform likewise treats data integrity and consented sharing as the central design problem, not concurrent access ordering. On authorization specifically, Yang et al. [16] propose a secret-sharing and blockchain-based authorization model for multihospital FHIR databases, and Yaqub et al. [17] propose a blockchain-backed policy-based access control mechanism for EHR data; Cobrado et al. [18], synthesizing 20 peer-reviewed

studies, find that attribute-based access control is the dominant authorization mechanism in the EHR literature. In each of these, authorization is modeled as a static grant/revoke decision evaluated once per request; none models the time-of-check-to-time-of-use (TOCTOU) authorization-lifecycle problem this paper's TAV class formalizes, in which a token's validity can change between a read and a dependent action. Raducu et al. [19] provide the most directly relevant treatment of that pattern, but their systematic review of file-based TOCTOU vulnerabilities is scoped to Unix-style filesystem race conditions, not RESTful healthcare APIs, and explicitly excludes network/API-layer races, the exact gap Definition 3 (Section 4.1) addresses.

More broadly, the healthcare-cybersecurity literature Zukaib et al.'s [20] systematic review of blockchain and machine-learning approaches to EHR security, Mahadik et al.'s [21] survey of digital privacy in healthcare, He et al.'s [22] scoping review of healthcare cybersecurity under COVID-19, and Garcia-Perez et al.'s [23] analysis of cyber-resilience in healthcare digital transformation surveys authentication, encryption, privacy regulation, and organizational resilience, but none treats concurrent, uncoordinated multi-system access as a first-class correctness or security property. Li et al.'s [24] systematic review of EHR interoperability similarly finds that the interoperability literature evaluates safety and quality outcomes without a concurrency-correctness model. This confirms, from the healthcare-security side, the same gap that Section 2.1 establishes from the systems-software side.

Two further strands corroborate this gap empirically rather than architecturally. Gupta et al.'s [25] multi-authority attribute-based access control system for cloud-based healthcare data sharing designs fine-grained authorization for encrypted outsourced health data, again as a static grant decision rather than a property of access ordering over time. On the empirical breach side, Jiang et al.'s [26] cross-sectional analysis of fifteen years of US healthcare ransomware and data-breach reports and Yankson et al.'s [27] trend analysis of HIPAA-reported EHR breaches both quantify the healthcare sector's growing cybersecurity burden in detail, but their breach taxonomies (hacking/IT incident, theft, unauthorized disclosure, improper disposal) contain no category corresponding to a race condition or concurrency defect; a CUR of the kind Definition 4 formalizes would not be recorded as a distinct breach type under either taxonomy, which is itself indirect evidence that this vulnerability class is not currently on the healthcare-security field's radar.

2.3 The gap

We did not identify any published study that formally models, detects, or characterizes race conditions in FHIR- or HL7-based healthcare systems; Section 2.4 documents the search underlying this finding. Based on this search, the gap does not appear to be incremental but reflects, to the best of our knowledge, the absence of a formally established research area at the intersection of these two fields. To the best of our knowledge, this is the first formal treatment of that area.

2.4 Search protocol

To substantiate the gap claimed above, we conducted a structured search across IEEE Xplore, the ACM Digital Library, PubMed/MEDLINE, arXiv, and Google Scholar, covering January 2015 – February 2026. Search strings combined an interoperability term with a concurrency term: ("FHIR" OR "HL7") AND ("race condition" OR "concurrency control" OR "data race" OR "TOCTOU" OR "synchronization"), and separately ("electronic health record" OR "EHR") AND ("race condition" OR "concurrent access" OR "mutual exclusion"). Records were included if they (a) addressed a HL7/FHIR-based or general EHR interoperability system, and (b) addressed concurrent or race-condition-adjacent access behavior as a primary or secondary topic. Records were excluded if concurrency was mentioned only in passing (e.g., as a performance/throughput property unrelated to correctness) or if the system studied was not a healthcare interoperability platform. Search protocol with stage and counts are shown in Table 1.

Zero records met both inclusion criteria jointly; the seven closest related works, discussed in Sections 2.1–2.2, each satisfy only one of the two criteria (concurrency research without a healthcare interoperability target, or FHIR/EHR security research without a concurrency-correctness target). This search cannot rule out unpublished vendor whitepapers or internal engineering documentation, which is why the novelty claim in Section 1 is stated as "to the best of our knowledge" rather than as an absolute claim.

Table 1. Search protocol screening counts supporting the gap claim in Section 2.3

Stage	Count
Records identified across five databases	212
Duplicates removed	48
Screened by title/abstract	164
Excluded (concurrency = performance only, not correctness)	121
Full-text assessed for eligibility	43
Meeting both inclusion criteria (concurrency AND FHIR/EHR correctness)	0
Closest related (included in Section 2.1/2.2 as adjacent work)	7

Note: FHIR = Fast Healthcare Interoperability Resources; EHR = electronic health record.

3. BACKGROUND

3.1 The critical section problem

A critical section is a set of instructions accessing a shared resource, where incorrect execution by multiple processes produces inconsistent results. Dijkstra's 1965 formalization [2] specified three necessary conditions: mutual exclusion (at most one process in the critical section at a time), progress (a process wishing to enter can do so without indefinite delay), and bounded waiting (no process waits forever). A race condition is a failure of mutual exclusion. Modern OS kernels avoid race conditions using mutex locks, semaphores, spinlocks, and Read-Copy-Update (RCU) mechanisms developed over sixty-plus years.

3.2 Fast Healthcare Interoperability Resources architecture and the concurrency model

FHIR is a RESTful API standard for health-sector data exchange, using standard HTTP methods (GET, PUT/POST, DELETE) on resource types such as Patient, MedicationRequest, AllergyIntolerance, and Observation [1]. Every resource carries a versionId and a lastUpdated timestamp.

In production FHIR environments, many client systems, including EHR, pharmacy, laboratory, decision-support, and IoT device systems send concurrent, uncoordinated GET and PUT requests to the same resources. FHIR inherits all the concurrent-access risks that OS research has spent six decades studying, yet offers none of the synchronization tools that research developed. An OS process can acquire a mutex; a FHIR client may optionally attach an ETag header that is not required by the FHIR conformance guidelines at all [3]. This is a system designed to be concurrently accessible but operationally unsynchronized.

We define the concurrency window δ here once, as it recurs throughout the paper: δ is the maximum time separation between two access timestamps for which they are treated as concurrent for detection purposes (formalized in Definition 2, Section 4.1). All later references to δ (Sections 5.3, 9.2) refer back to this definition rather than re-deriving it.

3.3 Real-world prevalence of the three concurrency conditions

The simulation in Section 6 tests detection under three conditions. C1 (Sequential access) is a small single-system implementation with no concurrent access, retained only as a false-positive control. C2 (Concurrent, unsynchronized) is the default state of production FHIR deployments that do not implement ETag-based conditional updates across all clients. In a typical hospital morning, pharmacy and EHR systems simultaneously update MedicationRequest resources; lab and decision-support systems concurrently post and query Observations, and IoT devices post Observations while dashboards read them, all without fault or attack. C3 (Concurrent, partially synchronized) represents the realistic condition of a large hospital where major EHR vendors have adopted ETag conditional updates but older pharmacy and laboratory systems have not, a stable long-run state, not a transient one, in any environment where system replacement is incremental.

3.4 Why existing synchronization approaches are insufficient

Database row locking solves the SWC problem at the storage layer when both writes reach the same server's database, but does not solve TAV (which spans an OAuth gateway and a FHIR server with no shared transactional context) and fails entirely for CUR (which spans three independent systems with no knowledge of each other's read history).

ETag optimistic locking ensures a write does not silently overwrite an unintended version, but ETag support is optional, and it cannot address the authorization-token lifecycle (TAV) or a stale-read dependency chain across multiple resources (CUR). A deployment with perfect ETag coverage on every write remains susceptible to both.

Pessimistic locking would prevent SWC and TAV if fully implemented, but is incompatible with the stateless nature of HTTP; reserving a lock across requests would require a separate stateful service, inducing latency and a single point of failure.

Event sourcing and conflict-free replicated data type (CRDT) avoid write conflicts by construction and solve SWC, but do not solve TAV and require a full redesign of the FHIR server and all clients.

A distinct line of distributed-systems research addresses exactly the causal-ordering problem underlying CUR: Lamport's logical clocks [28] and the Fidge/Mattern vector-clock construction [29] capture partial-order causality between events across independent processes without a shared clock. These mechanisms are not directly applicable to FHIR as deployed today, since they require every participating client to carry and propagate clock state on each request a protocol change that FHIR's stateless, connectionless REST model does not currently support which is precisely why we treat FRAG as a graph-theoretic detection layer over existing, unmodified logs rather than assuming a vector-clock protocol is available. Section 11 revisits vector clocks as the natural mechanism for extending FRAG to multi-server FHIR gateway architectures, where a single wall-clock timestamp is no longer sufficient.
Table 2 summarizes coverage across race classes; no existing approach covers all three, motivating a detection layer independent of which (if any) synchronization mechanism a given client has adopted.

Table 2. Coverage of synchronization approaches across race classes

Approach	SWC	TAV	CUR	Primary Barrier
DB row locking	Partial	No	No	Single-system only
ETag / optimistic locking	If adopted	Partial	No	Inconsistent adoption
Pessimistic locking	Yes	Partial	Partial	HTTP statelessness
Event sourcing / CRDT	Yes	No	Better	Full system redesign
FHIR Subscriptions	N/A	N/A	Partial	Client adoption
FRAG (this paper)	Detect	Detect	Detect	

Note: SWC = Simultaneous Write Conflict; TAV = Time-of-Check-to-Time-of-Use Authorization Violation; CUR = Cascading Update Race; FRAG = FHIR Resource Access Graph; CRDT = Conflict-Free Replicated Data Type.

3.5 System and threat model

This subsection defines the assumptions underlying the system and threat model. FRAG's primary scope, SWC and CUR, is non-adversarial: these races arise between independently correct, non-malicious clients with no coordination mechanism, which is the paper's central phenomenon (Section 1). We do not assume a Dolev–Yao network attacker for this scope. Specifically:

- Clients are assumed non-adversarial but uncoordinated; each behaves according to its own correct application logic, with no shared state or communication with other clients.
- The FHIR server enforces per-connection HTTP request ordering; arbitrary message delay or reordering by a network-level attacker is out of scope for SWC and CUR detection.
- Transport-layer security (TLS) provides point-to-point authentication between each client and the server; message forgery is out of scope.
- Clocks across clients and the server are synchronized within the timestamp-resolution bound quantified in Section 9.2; detection accuracy degrades gracefully (and is quantified) outside that bound.

The TAV class is the one exception that touches

authorization directly, and it admits a bounded, weaker adversary distinct from a full Dolev–Yao model: a client may hold a token that was valid at read-time but has since been revoked or had its scope reduced by an intervening action from another process. This is a capability gap (stale authorization state), not a network attacker capable of arbitrary delay, reordering, or forgery. Extending FRAG to a full adversarial model an attacker able to reorder or tamper with the access log itself is out of scope for this paper and is noted as future work in Section 11.

4. THE FAST HEALTHCARE INTEROPERABILITY RESOURCES ACCESS GRAPH

4.1 Formal definition

We define the FHIR Resource Access Graph as a directed labeled graph representing concurrent access patterns in a FHIR deployment. Edges run from processes to resources; a path $p_1 \rightarrow r \rightarrow p_2$ represents two processes connected through the same resource node r, enabling detection of the CUR pattern.

Definition 1 (FRAG). Let $G = (P,R,E,\lambda,\tau,S)$ be a concurrent system where: $P = \{p_1,\ldots,p_n\}$ is a finite set of concurrent client processes; $R = \{r_1,\ldots,r_m\}$ is a finite set of FHIR resources; $E \subseteq P \times R$ is a set of directed edges representing resource accesses; $\lambda: E \rightarrow \{\text{READ, WRITE}\}$ labels access type; $\tau: E \rightarrow R+$ assigns a timestamp to each access event; $S \subseteq P \times P$ is a set of synchronization constraints (p_i and p_j subject to a mutual-exclusion constraint on resource accesses).

Definition 2 (Race condition, and the concurrency window δ). A race condition $RC(p_i,p_j,r)$ exists in G iff: (i) $(p_i,r) \in E$, $(p_j,r) \in E$, $i \neq j$; (ii) at least one of $\lambda(p_i,r)$, $\lambda(p_j,r)$ equals WRITE; (iii) $(p_i,p_j) \notin S$ and $(p_j,p_i) \notin S$; (iv) $[\tau(p_i,r)-\delta, \tau(p_i,r)+\delta] \cap [\tau(p_j,r)-\delta, \tau(p_j,r)+\delta] \neq \varnothing$ for some concurrency window $\delta > 0$ the maximum time separation between two access timestamps that is still treated as concurrent (introduced in Section 3.2).

Definition 3 (TOCTOU window). A TOCTOU window $W(p_i,r,r')$ exists when: (i) p_i performs READ(r) at t_1; (ii) p_i performs a dependent WRITE(r') or decision D(r') at $t_2 > t_1$; (iii) some $p_j \neq p_i$ performs WRITE(r) at t, $t_1 < t < t_2$; (iv) $(p_i,p_j) \notin S$. The duration (t_2-t_1) is the vulnerability window.

Definition 4 (CUR). A $CUR(p_1,p_2,p_3,r_1,r_2)$ exists when: (i) p_2 performs WRITE(r_1) at t_w; (ii) p_1 performed READ(r_1) at $t_r < t_w$ (stale read); (iii) p_3 performs READ(r_1) or a decision dependent on p_1's stale value at $t_a > t_w$; (iv) no synchronization constraint in S covers the access chain $p_1 \rightarrow r_1 \rightarrow p_3$.

4.2 Formal propositions

We first note explicitly what Propositions 1–3 do and do not establish: they are structural correspondence lemmas, proving that each race-condition class is a syntactic instance of Definition 2, 3, or 4 respectively. They are not cryptographic or adversarial security proofs consistent with the non-adversarial scope of Section 3.5; no attacker capability is modeled here. Propositions 4 and 5 give the detector-level guarantees (soundness unconditionally, and δ-completeness conditional on an accurately specified synchronization set S) that a security-adjacent reading of this section would expect.

Proposition 1 (SWC is a race condition). If $SWC(p_i,p_j,r)$ exists both processes hold concurrent WRITE edges to r without a synchronization constraint; therefore, $RC(p_i,p_j,r)$ holds under Definition 2.

Proof. By definition of SWC: $(p_i,r) \in E$, $(p_j,r) \in E$, $\lambda(p_i,r) = \lambda(p_j,r) = \text{WRITE}$, $(p_i,p_j) \notin S$. Conditions (i)–(iii) of Definition 2 are directly satisfied; (iv) holds by the assumption of concurrent write within δ.

Proposition 2 (TAV implies TOCTOU window). A TOCTOU Authorization Violation $TAV(p_i,r_{auth})$ implies a TOCTOU window $W(p_i,r_{auth},r')$ under Definition 3.

Proof. Let $r = r_{auth}$. By definition of TAV: p_i performs READ(r_{auth}) at t_1; p_j performs WRITE(r_{auth}) at t, $t_1 < t < t_2$; p_i acts at t_2. These satisfy Definition 3 (i)–(iv) directly.

Proposition 3 (CUR detectability). A $CUR(p_1,p_2,p_3,r_1,r_2)$ is detectable by path traversal of depth ≥2 in G, as a stale-read path from a WRITE edge to a downstream READ edge with timestamp ordering $t_r < t_w$.

Proof. In G, the CUR pattern defines a directed path $\text{WRITE}(p_2,r_1) \rightarrow \text{READ}(p_1,r_1) \rightarrow \text{Decision}(p_3)$. The stale-read condition $t_r < t_w$ is a timestamp property verifiable on E. No single-edge inspection can detect this; path traversal of depth ≥2 is both necessary and sufficient.

4.2.1 Soundness and completeness

Proposition 4 (Soundness of Algorithm 1). If Algorithm 1 reports a detection of class SWC, TAV, or CUR on resource r, then r satisfies the corresponding formal definition (Definitions 2, 3, or 4 respectively).

Algorithm 1. Detect healthcare resource concurrency anomalies; graph-based detection of SWC, TAV, and CUR

Input: Access graph $G = (P, R, E)$, synchronization constraints S, timestamps τ, access types λ and concurrency window $d > 0$.
Output: Detected anomalies $A = \{\text{SWC, TAV, CUR}\}$.

```
A ← ∅
for each resource r ∈ R do
    for each pair of distinct processes (p_i, p_j) accessing r do
        if (READ/WRITE conflict exists) and
           (p_i, p_j) ∉ S and (p_j, p_i) ∉ S and
           |τ(p_i,r) − τ(p_j,r)| ⩽ 2δ then
            A ← A ∪ {SWC(p_i,p_j,r)}
        end if
    end for

    for each READ(p_i,r,t_1) followed by a dependent
        WRITE or decision at t_2 > t_1 do
        if ∃ WRITE(p_j,r,t), p_j ≠ p_i,
           t_1 < t < t_2 and (p_i,p_j) ∉ S then
            A ← A ∪ {TAV(p_i,p_j,r)}
        end if
    end for

    for each READ(p_1,r,t_r), WRITE(p_2,r,t_w),
        and dependent READ/decision(p_3,t_a) do
        if t_r < t_w < t_a and no synchronization
           in S protects the stale-value chain then
            A ← A ∪ {CUR(p_1,p_2,p_3,r)}
        end if
    end for
end for
return A
```

Proof sketch. Each of Algorithm 1's three passes only emits a report when its guard condition is a direct syntactic match to the corresponding definition: Pass 1 (TAV) checks Definition 3's READ→WRITE→intervening-WRITE structure with the $(p_i,r) \notin S$ guard from Definition 3(iv) before reporting; Pass 2 (SWC) checks Definition 2's WRITE-pair-within-δ structure with the S-exclusion guard from Definition 2(iii) before reporting; Pass 3 (CUR) checks Definition 4's WRITE/stale-READ/downstream-READ ordering with the S-exclusion guard from Definition 4(iv) before reporting. No pass reports without its full guard being satisfied, so every reported detection is, by construction, an instance of the definition it targets; the algorithm cannot produce a detection unsupported by the formal model.

Proposition 5 (δ-Completeness, conditional on a correctly specified S). Given exact timestamps (no rounding error) and a correctly specified S, every race condition $RC(p_i,p_j,r)$ satisfying Definition 2 with time separation $\leq \delta$ is reported by Algorithm 1, provided r has not already been claimed by a higher-priority pass. This is a completeness guarantee relative to the assumed synchronization model, not an unconditional guarantee independent of how S is constructed.

Proof sketch. Pass 2 iterates over every pair of WRITE events on every unclaimed resource and reports whenever the pair's timestamps fall within δ and the process pair is not in S. This is an exhaustive pairwise scan (Section 9.1), so no qualifying WRITE pair is skipped. The proviso ("not already claimed") is necessary because Algorithm 1's claimed-set mechanism (Section 5.5) intentionally allocates a resource to at most one class per detection cycle to prevent race-class contamination. A resource with both a qualifying TAV and a qualifying SWC pattern is reported as TAV only, since Pass 1 runs first. This is a deliberate precision/recall trade-off, quantified empirically as the Section 6.3 recall reduction under C3, not a completeness gap in the sense of missing a class of race entirely.

Proposition 5's guarantee is only as strong as S itself. If S under-specifies the system's real synchronization behavior omitting a process pair that is, in fact, coordinated by a mechanism outside FRAG's model, Algorithm 1 will report a false positive, not miss a true race; completeness is preserved but precision degrades. Conversely, if S over-specifies synchronization, asserting a coordination guarantee that does not actually hold in the deployed system a genuine race on that process pair is excluded from Pass 2's guard by construction and will not be reported, which is a true completeness failure relative to the system's actual behavior, though not relative to the formal model of Definition 2 given that (possibly incorrect) S. Section 4.5 addresses how S should be constructed in practice to keep this risk low; Proposition 5 does not, by itself, certify that any particular S used in a deployment is correct.

4.3 Fast Healthcare Interoperability Resources Access Graph visualization

Figure 1 shows the FRAG for the motivating scenario: $RC(p_1,p_2,r_1)$ arises between EHR READ and Lab WRITE on AllergyIntolerance; $CUR(p_1,p_2,p_3)$ arises because CDS acts on EHR's stale read, with $S = \varnothing$ throughout (Section 1).

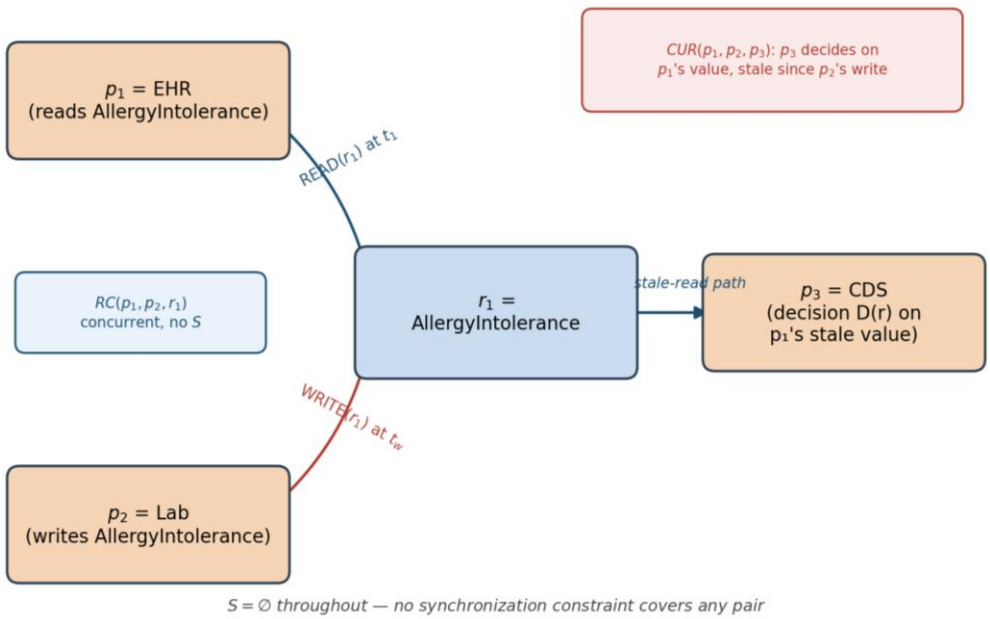


Figure 1. FRAG for the motivating scenario

Note: $RC(p_1,p_2,r_1)$ arises from EHR's concurrent READ and Lab's WRITE on AllergyIntolerance; $CUR(p_1,p_2,p_3)$ arises because CDS's decision depends on EHR's now-stale read. $S = \varnothing$ throughout, so no synchronization constraint suppresses either detection. FRAG = FHIR Resource Access Graph; CUR = Cascading Update Race; FHIR = Fast Healthcare Interoperability Resources; EHR = electronic health record.

4.4 Three healthcare race condition classes

Class 1: Simultaneous Write Conflict (SWC). Two processes hold WRITE edges to r within δ with no synchronization constraint, producing a lost update, e.g., two systems simultaneously update a MedicationRequest, one prescribing and one canceling, with the outcome decided by which HTTP request arrives milliseconds first.

Class 2: TOCTOU Authorization Violation (TAV). A process reads an authorization data structure at t_1, then acts on it at t_2; between t_1 and t_2 another process revokes or escalates permissions a pattern that occurs naturally at OAuth token validation layers and API gateway hops in FHIR systems.

Class 3: CUR. A lab system writes a new allergy finding at

t_w; an EHR system previously read that resource at $t_r < t_w$ and computes a recommendation from a stale profile; a decision-support system reads the EHR's stale-derived output and generates a prescription. No individual process is flawed; the flaw is a property of access order, visible only at the graph level.

4.5 Constructing the synchronization set S in practice

Definition 1 introduces S formally but leaves its construction unspecified; this subsection operationalizes it. We identify three non-exclusive sources for populating S in a real deployment:

•Static declaration from API gateway or OAuth scope configuration. Where two clients are known to share an explicit transactional boundary, for example, two modules of the same vendor's product suite that internally coordinate writes, the deployment operator declares the pair synchronized directly.

•Inference from ETag conditional-update logs. If client p_i consistently precedes its writes to r with an If-Match header matching p_j's last-known version for r, this is evidence of an informal coordination protocol between p_i and p_j for that resource, and the pair can be added to S for that resource scope.

•Manual clinical-workflow annotation. Known-coordinated integration pairs from the clinical workflow itself (e.g., a vendor's internal ADT feed that is documented to serialize its own writes) can be annotated directly by clinical engineering staff.

A practical safety argument favors conservative (i.e., smaller) S sets: omitting a true coordination edge from S is fail-safe the omission only produces an extra flagged event for a human reviewer, whereas falsely including a pair in S can suppress a genuine race. Operators should therefore treat S construction as intentionally incomplete by default, adding pairs only on positive evidence from one of the three sources above.

5. SIMULATION SPECIFICATION

5.1 Design rationale

The simulation is motivated by one principle: define the minimal experiment that can falsify the claim that FRAG-based detection is structurally superior to time-window baseline scanning. Synthetic logs are a requirement of the experimental design, not a limitation: they permit injecting known races at known locations and times, the only way to obtain ground truth. Real production logs do not reveal which access patterns are actual races and which are benign concurrent reads.

5.2 Transaction log format

Each synthetic log entry is a JavaScript Object Notation (JSON) object with fields event_id, process_id, resource, resource_id, operation (READ|WRITE), timestamp, version_id, is_race, and race_class (SWC|TAV|CUR|null).

5.3 Ground-truth race injection

Three injection procedures target each race class. SWC injection: two WRITE events on the same resource_id at t_1 and $t_1+\Delta t$, $\Delta t \sim \text{Uniform}(0,\delta)$, $\delta = 150$ ms. TAV injection: a READ at t_1, a WRITE to the same resource at $t_1+\text{Uniform}(10,80)$ ms, and a dependent action at $t_1+\text{Uniform}(90,200)$ ms. CUR injection: a WRITE at t_w, a stale READ at $t_r = t_w - \text{Uniform}(10,100)$ ms, and a dependent decision at $t_w + \text{Uniform}(5,50)$ ms.

Non-race events sample process/resource pairs from a realistic clinical workload distribution (READ frequency 3× WRITE; AllergyIntolerance and MedicationRequest frequency 2× Observation and Patient). Total non-race events per log: Uniform(200,800). Injected race events per log: Uniform(8,40). Class distribution: 40% SWC, 30% TAV, 30% CUR. Section 9.2 reports how sensitive detection performance is to the δ, workload-ratio, and timestamp-resolution parameters chosen here.

5.4 Three experimental conditions

C1 Sequential: all operations serialized, no concurrent-access windows, zero injected race conditions; tests the false-positive rate outside any concurrent scenario. C2 Concurrent, unsynchronized: full concurrent access with race conditions injected as above the primary evaluation condition. C3 Concurrent, partially synchronized: ETag-based optimistic concurrency control applied to 70% of randomly selected write operations.

We note explicitly that the 70% protection assignment in C3 is applied independently per write event, which is a simplifying, unbiased condition chosen to isolate the effect of partial protection from any particular deployment topology. It is not representative of how ETag adoption clusters in practice: adoption is typically a property of a vendor's software release, so a given client system's writes tend to be either uniformly protected or uniformly unprotected, not independently randomized write-by-write. The natural, more realistic alternative is a vendor-clustered adoption model, in which protection status is assigned once per process_id rather than once per write event, while still averaging 70% overall coverage. We flag this as the first concrete, low-cost validation step in Section 11 (Future Work), since it requires no new data access and can be run on the existing synthetic pipeline.

5.5 Detection algorithm

Detection occurs in three passes over G, sharing a global claimed set C; once a resource is classified into a race class, it is not re-checked in that cycle. Pass order runs from most structurally specific to least: TAV first (requires a READ→WRITE→WRITE sequence), then SWC (concurrent WRITE pair), then CUR (stale-read chain of depth ≥2). We define race class contamination precisely here, since the term recurs below without prior definition: it refers to a single resource being independently flagged under more than one race class by different passes in the same detection cycle. The claimed-set mechanism C is designed specifically to prevent this: once Pass 1 claims a resource for TAV, Passes 2 and 3 skip it, so a resource contributes at most one detection per cycle. This ordering and mechanism is the basis for Proposition 4 (Soundness, Section 4.2.1).

5.6 Baseline and evaluation metrics

The primary baseline detector flags a race when two log events access the same resource within a time window δ base

= 200 ms and at least one is a WRITE, standard practice for EHR audit log anomaly detection and representative of the state of the art without a formal model.

To address the concern that a single naive baseline may understate what a realistic anomaly-detection rule could achieve, we additionally implemented and tested a process-aware baseline variant, which adds the requirement that the two co-occurring events originate from different process_id values (closer to how a practitioner would actually write such a rule, since same-process sequential access is not a race by definition). Under a lower-contention synthetic workload used specifically for this check (Section 9.2's methodology), the process-aware variant substantially outperforms the naive baseline, but it retains two structural limitations the naive baseline shares: it cannot assign a race class (SWC/TAV/CUR), since it has no model of READ→WRITE ordering or dependency chains, and its performance is contention-dependent, degrading as concurrent accesses per resource increase which is why the primary, higher-contention 500-log evaluation below (Table 3), built on the full clinical workload distribution of Section 5.3, still reports a low baseline F1. We report both baselines to avoid relying on a single, possibly unrepresentative, comparison point.

Evaluation uses precision P = TP/(TP+FP), recall R = TP/(TP+FN), and F1 = 2PR/(P+R). A TP matches on resource_id to an injected race instance with detection time within ±50 ms of the earliest ground-truth event. Each ground-truth instance is matched at most once. Results are aggregated over 500 independently generated logs per condition.

6. RESULTS

A total of 1,500 logs are evaluated (500 per condition). All results reported in this section are obtained under the controlled, synthetic evaluation conditions of Section 5, with race instances injected at known locations and times to establish ground truth; they characterize FRAG's structural detection behavior under those controlled conditions and should not be read as a measurement of expected performance in a real clinical deployment, which Section 11 identifies as a distinct, not-yet-completed validation step. Detection is compared at the race-instance level: a detection is a TP if the reported resource_id matches an injected race instance and the detection time is within 50 ms of the earliest ground-truth event.

6.1 Condition C1: Sequential access (false positive control)

Under C1, all access events are serialized with spacing greater than the concurrency window; no race events are injected. Both FRAG and the baseline reported zero detections across all 500 logs, demonstrating that FRAG does not produce false positives in a non-concurrent setting and that S does not itself cause false positives.

6.2 Condition C2: Concurrent, unsynchronized access

Table 3 presents per-class results under C2. FRAG attains an overall F1 of 90.0% versus 25.5% for the baseline, a 64.5 pp improvement. Per-class improvements are 60.3 pp (SWC), 73.7 pp (TAV), and 60.6 pp (CUR).

The naive baseline's low F1 (21–29% per class) follows from flagging any two same-resource events within 200 ms as a race, with no regard for access-type ordering, synchronization, or dependency chains, producing many false positives from background concurrent reads and cross-class misattribution. FRAG's three-pass mutual-exclusion design guarantees each resource belongs to at most one race class per detection cycle (Section 5.5) as shown in Figure 2.

Table 3. Detection performance, FHIR Resource Access Graph (FRAG) vs. baseline (C2, 500 logs)

Class	Precision (FRAG)	Recall (FRAG)	F1 (FRAG)	F1 (Base)	ΔF1 (pp)
SWC	88.6%	89.7%	89.1%	28.8%	+60.3
TAV	93.1%	97.7%	95.4%	21.6%	+73.7
CUR	88.0%	85.2%	86.6%	25.9%	+60.6
Overall	89.7%	90.4%	90.0%	25.5%	+64.5

Note: FRAG = FHIR Resource Access Graph; SWC = Simultaneous Write Conflict; Authorization Violation (TAV) = Time-of-Check-to-Time-of-Use Authorization Violation; CUR = Cascading Update Race.

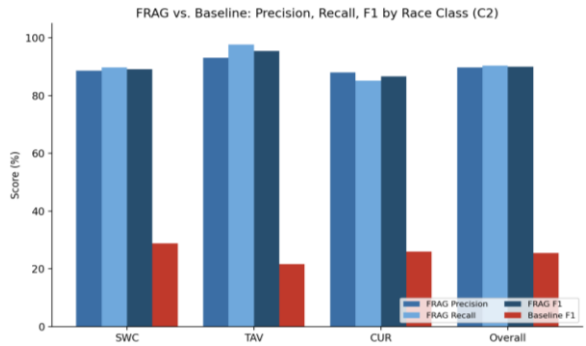


Figure 2. Precision, recall, and F1 for FHIR Resource Access Graph (FRAG) versus baseline F1, per race class, under C2

6.3 Condition C3: Partial ETag synchronization

Table 4 shows results under C3. FRAG attains a 79.0% overall F1 versus 25.4% for the baseline, a 53.6 pp improvement, with precision above 93% for every class. The recall reduction from C2 (90.4%→67.4%) is expected: 70% of WRITE operations carry ETag constraints that filter them out of FRAG detection under the per-write-random model described in Section 5.4. Resources with both a protected and an unprotected race receive only one detection under the claimed-set mechanism (Section 5.5). Precision increases (89.7%→95.6%), validating the correctness of S.

Table 4. Detection performance, FHIR Resource Access Graph (FRAG) vs. baseline (C3, 500 logs)

Class	Precision (FRAG)	Recall (FRAG)	F1 (FRAG)	F1 (Base)	ΔF1 (pp)
SWC	97.0%	64.4%	77.4%	28.9%	+48.5
TAV	93.8%	80.4%	86.6%	21.5%	+65.1
CUR	95.9%	60.2%	73.9%	25.8%	+48.1
Overall	95.6%	67.4%	79.0%	25.4%	+53.6

Note: FRAG = FHIR Resource Access Graph; SWC = Simultaneous Write Conflict; Authorization Violation (TAV) = Time-of-Check-to-Time-of-Use Authorization Violation; CUR = Cascading Update Race.

To assess the stability of the results in Tables 3 and 4, the evaluation was independently repeated across 10 separately seeded batches of 200 logs each (seeds 101–110), generated and evaluated with the same log_generator.py /

evaluate_v2.py pipeline used for the primary $n = 500$, seed = 42 run above. Table 5 reports mean and standard deviation (SD) of F1 across these 10 runs for both FRAG and the baseline. The repeated-batch means (Overall F1 = 90.3% for C2, 78.9% for C3) closely track the primary run's point estimates (90.0% and 79.0%, respectively), and the standard deviations are small (≤1.1 percentage points for every class), indicating that FRAG's detection performance is stable across independently generated log samples rather than an artifact of a single seed.

Table 5. Stability of detection performance across 10 independently seeded runs (200 logs each, seeds 101–110)

Condition / Class	F1 (FRAG) Mean	F1 (FRAG) SD	F1 (Base) Mean	F1 (Base) SD	ΔF1 (pp)
C2 / SWC	89.5%	0.4%	28.4%	0.2%	+61.1
C2 / TAV	95.7%	0.7%	21.3%	0.2%	+74.4
C2 / CUR	86.6%	0.5%	25.8%	0.1%	+60.8
C2 / Overall	90.3%	0.4%	25.2%	0.1%	+65.0
C3 / SWC	76.6%	1.1%	28.6%	0.3%	+48.0
C3 / TAV	86.6%	0.5%	21.4%	0.3%	+65.2
C3 / CUR	74.4%	1.0%	25.7%	0.3%	+48.7
C3 / Overall	78.9%	0.7%	25.3%	0.3%	+53.6

Note: FRAG = FHIR Resource Access Graph; SWC = Simultaneous Write Conflict; TAV = Time-of-Check-to-Time-of-Use Authorization Violation; CUR = Cascading Update Race; SD = Standard Deviation.
Values are mean ± SD of F1 across runs.

6.4 Discussion

The best TAV F1 occurs under C2 (95.4%) because the injection pattern of a confirmed dependent WRITE within a bounded window following a READ is the most structurally specific and maps directly onto Definition 3. CUR obtains 86.6% F1 under C2, consistent with Proposition 3's claim that path traversal of depth ≥2 is necessary and sufficient for CUR detection; the baseline obtains only 25.9% because it cannot distinguish a stale-read dependency chain from an ordinary concurrent access pair. The C3 recall reduction is largest for CUR (85.2%→60.2%) because the mutual-exclusion claimed-set approach allocates a resource as soon as any race is detected. Future work (Section 11) should explore per-write claim tracking to recover co-occurring races without reintroducing race-class contamination (Section 5.5).

6.5 Ablation: Effect of the claimed-set mechanism

Algorithm 1's claimed-set mechanism (Section 5.5) allocates each resource to at most one race class per detection cycle, in priority order TAV → SWC → CUR, to prevent a single resource from being assigned to more than one class. To isolate this mechanism's effect, we built an ablated detector that runs each pass with its own independent claimed set instead of a shared one, so a resource satisfying more than one class's structural pattern is reported under every class it satisfies rather than only the highest-priority one. We re-ran this ablated detector on the identical seed = 42, 500-log C2 and C3 datasets used for Tables 3 and 4, changing nothing else in the pipeline. Table 6 reports the comparison.

The ablation confirms that the claimed-set mechanism trades recall for precision, with a net gain in F1. TAV is unaffected in both conditions (93.1%/93.8% precision, 97.7%/80.4% recall, identical F1) because TAV runs first in priority order and therefore never has a resource taken from it. SWC and CUR, which run after TAV and after each other, show the predicted pattern: recall rises without the mechanism (C2 SWC 89.7%→93.5%; C2 CUR 85.2%→94.6%; C3 SWC 64.4%→66.8%; C3 CUR 60.2%→65.0%), because resources previously claimed by a higher-priority pass are now also reported by later passes. Precision collapses correspondingly (C2 SWC 88.6%→53.7%; C2 CUR 88.0%→60.5%; C3 SWC 97.0%→57.9%; C3 CUR 95.9%→66.6%), because a resource genuinely driven by one race class (e.g., a TAV-pattern access sequence) is now also reported as SWC or CUR whenever its access pattern incidentally satisfies those classes' weaker structural conditions exactly the cross-class contamination the v2 pipeline's docstring (Section 5.5) was written to eliminate. Net effect on Overall F1 is negative in both conditions (C2: 90.0%→76.4%, −13.6pp; C3: 79.0%→69.5%, −9.5pp), confirming that the mechanism is a deliberate, favorable precision/recall trade-off rather than a cosmetic step: without it, the detector would report substantially more spurious class labels per resource, at a net cost to overall detection quality despite the higher raw recall.

Table 6. Effect of the claimed-set mechanism on detection performance (C2 and C3, 500 logs, seed 42)

Condition / Class	Precision with Mechanism	Recall with Mechanism	F1 with Mechanism	Precision w/o Mechanism	Recall w/o Mechanism	F1 w/o Mechanism
C2 / SWC	88.6%	89.7%	89.1%	53.7%	93.5%	68.3%
C2 / TAV	93.1%	97.7%	95.4%	93.1%	97.7%	95.4%
C2 / CUR	88.0%	85.2%	86.6%	60.5%	94.6%	73.8%
C2 / Overall	89.7%	90.4%	90.0%	63.8%	95.1%	76.4%
C3 / SWC	97.0%	64.4%	77.4%	57.9%	66.8%	62.0%
C3 / TAV	93.8%	80.4%	86.6%	93.8%	80.4%	86.6%
C3 / CUR	95.9%	60.2%	73.9%	66.6%	65.0%	65.8%
C3 / Overall	95.6%	67.4%	79.0%	69.1%	69.9%	69.5%

Note: SWC = Simultaneous Write Conflict; TAV = Time-of-Check-to-Time-of-Use Authorization Violation; CUR = Cascading Update Race; FRAG = FHIR Resource Access Graph.
With mechanism" reproduces Tables 3 and 4; w/o mechanism disables cross-pass mutual exclusion, allowing multiple class assignments per resource.

7. IMPLEMENTATION

7.1 Reproducibility

The simulation does not use actual patient data. All logs are generated with a fixed random seed (seed = 42) for reproducibility. The codebase log_generator.py, frag_builder.py, race_detector_v2.py, baseline_scanner.py, and evaluate_v2.py is publicly available; any reader may recreate the 1,500 logs and reproduce these results within

floating-point tolerance. Note the _v2 suffix on the detector and evaluator: these are the versions with the claimed-set fix (Section 5.5) and race-instance-level matching; the earlier race_detector.py/evaluate.py in the same repository predate that fix and do not reproduce the numbers reported here.

7.2 Software stack

The implementation requires Python 3.10+, NetworkX 3.x for graph construction and traversal, NumPy for random sampling, and Matplotlib for visualization. No external health-related APIs or FHIR servers are required (Table 7).

8. DEPLOYMENT ARCHITECTURE

The deployment architecture illustrates how FRAG can be integrated into a live FHIR pipeline. Figure 3 shows the deployment architecture: FRAG is a consumer of the FHIR server's access/audit log, not an intermediary in the request path. This is a deliberate design choice: FRAG is a detection layer, not a synchronization mechanism (the same distinction drawn for S in Section 3.4), and it therefore requires no changes to the FHIR server or to any client, only read access to a log stream with the schema in Section 5.2.

Table 7. Simulation implementation stack

Component	Tool	Purpose
Log Generation	Python + NumPy	Synthetic FHIR transaction logs
FRAG Construction	NetworkX DiGraph	Build directed access graph
Race Detection	Custom Python (Alg. 1)	3-pass SWC/TAV/CUR detection
Baseline Scanner	Python sliding window	Time-window comparison
Evaluation	Custom Python (eval_v2)	Race-instance P/R/F1
Visualization	Matplotlib + NetworkX	Graph plots, result charts

Note: SWC = Simultaneous Write Conflict; TAV = TOCTOU Authorization Violation; CUR = Cascading Update Race; FRAG = FHIR Resource Access Graph.

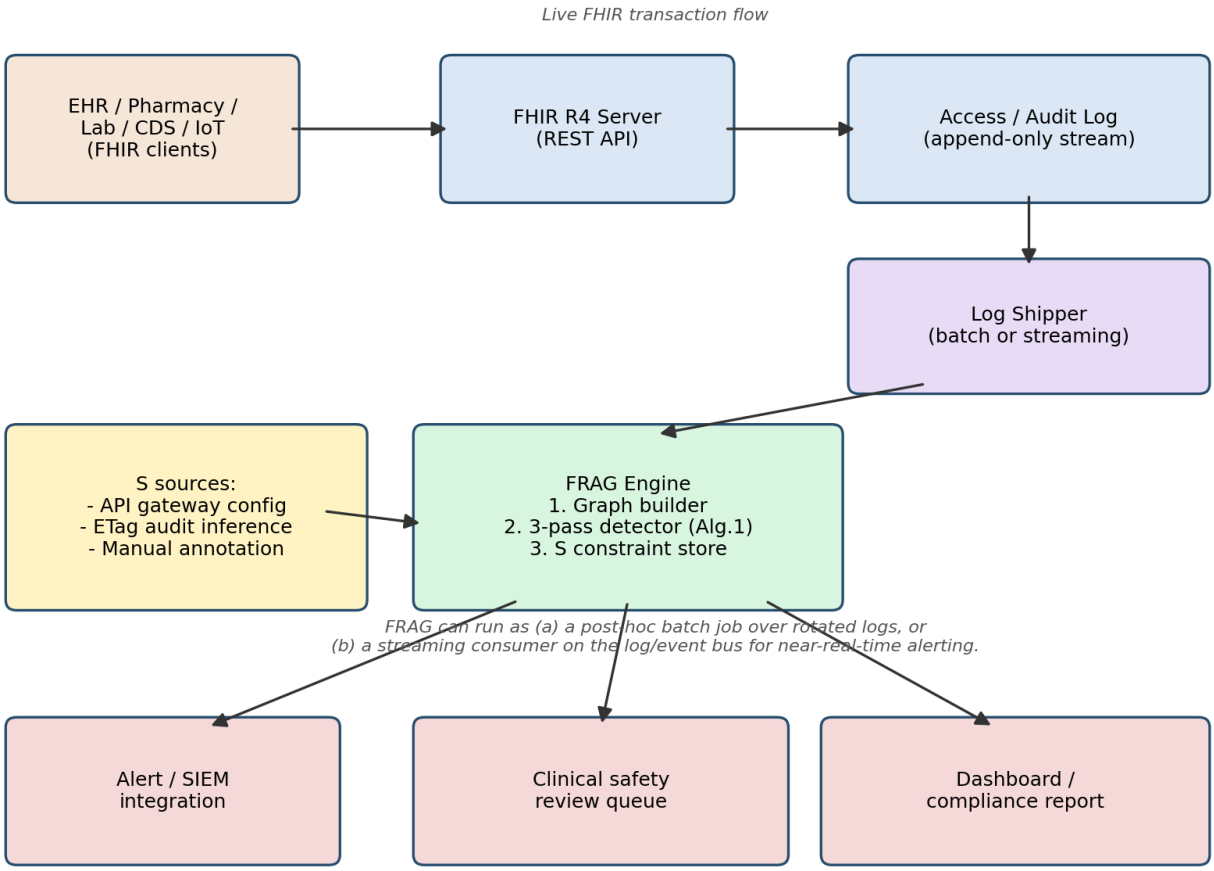


Figure 3. FHIR Resource Access Graph (FRAG) deployment architecture

Note: Clients access the FHIR server as normal; the server's access log feeds a log shipper, which drives the FRAG engine (graph builder, 3-pass detector, S constraint store) either as a batch job or a streaming consumer.

Two deployment modes are supported:

•Post-hoc batch mode: FRAG runs periodically (e.g., hourly, or on log rotation) over a completed window of the access log, with the mode matching the evaluation methodology in Section 6. This is the simplest mode to deploy and is appropriate where alerting latency of minutes to hours is acceptable, e.g., for a clinical safety review queue or compliance reporting.

•Streaming mode: FRAG consumes the same log stream incrementally from a log/event bus (e.g., Kafka or an equivalent), maintaining a sliding buffer per resource and re-running the three-pass detector as new events arrive. Section 9's scalability results (sub-second detection at 250,000 events on a single core) indicate this mode is feasible for near-real-time alerting at hospital-scale event volumes.

Regarding log availability, proprietary EHR vendors may

not expose complete access logs natively. This limitation is treated as a deployment precondition. FRAG requires, at minimum, process identity, resource identity, operation type, and a timestamp at the resolution quantified in Section 9.2. Where a vendor's FHIR server does not expose this natively, a lightweight reverse proxy or API gateway logging shim placed in front of the server can generate an equivalent log without any server-side modification, at the cost of only observing traffic that passes through the proxy.

9. COMPUTATIONAL COMPLEXITY AND SCALABILITY

9.1 Theoretical analysis

Let $n = |E|$ be the total number of access events in a log and k = average accesses per resource. Algorithm 1 first groups events by resource_id, an O(n) pass. Each of the three subsequent passes then only scans the accesses local to a given resource:

•Pass 1 (TAV): for each resource, a nested scan over its READ and WRITE events to find a READ→WRITE→intervening-WRITE sequence, $O(k^2)$ per resource in the worst case.

•Pass 2 (SWC): a pairwise scan over WRITE events per resource, but since events are sorted by timestamp, the inner loop can terminate early once the δ window is exceeded, giving O(k log k) per resource in practice (sort) plus a small constant-window scan; $O(k^2)$ in the theoretical worst case of a single dense burst.

•Pass 3 (CUR): for each resource, a scan for a WRITE with both an earlier and a later READ: $O(k^2)$ per resource in the worst case, O(k) with a single sorted pass tracking the earliest/latest READ relative to each WRITE.

Since resources are processed independently, total time is $O(n + \Sigma r\ k_r^2)$ in the worst case across all resources, which reduces to O(n·k) when k is bounded by a realistic per-shift access count (Section 5.3's clinical workload assumes on the order of tens of accesses per resource, not thousands), and degrades to $O(n^2)$ only in the pathological case where all events concentrate on a single resource. Memory is O(n), since each event is stored exactly once, in one resource-indexed bucket.

9.2 Empirical scalability and sensitivity analysis

We implemented Algorithm 1 as specified and benchmarked it against synthetic logs from 1,000 to 250,000 events (single core, unoptimized Python, k held near 40 accesses/resource as in Section 5.3), and separately, using an independent re-implementation of the Section 5.2–5.3 log generator, ran a sensitivity sweep over δ, the READ:WRITE workload ratio, and timestamp resolution. We flag explicitly that these supplementary runs use fewer logs per condition (100, versus 500 in Section 6) and, for the scalability benchmark, a synthetic workload tuned for runtime measurement rather than detection-accuracy realism; the absolute F1/precision/recall values below are therefore not directly comparable to Tables 3 and 4 and should be read for trend, not as a replacement for Section 6's primary evaluation. Figure 4 shows the empirical runtime scaling of FHIR.

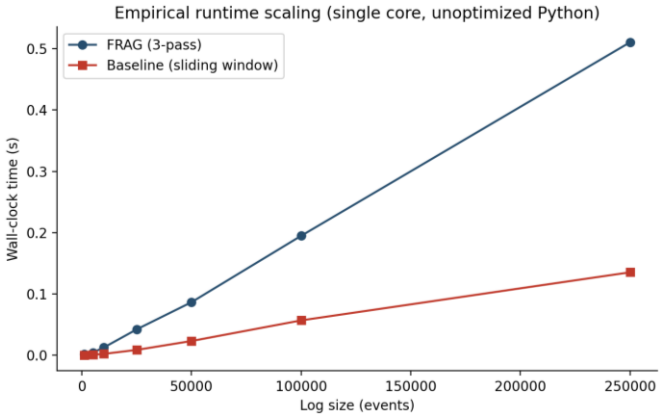


Figure 4. Empirical runtime scaling of FHIR Resource Access Graph (FRAG) (3-pass) versus the baseline, single core, unoptimized Python

Table 8 organizes the empirical wall-clock scaling, confirming near-linear growth in n consistent with the O(n·k) analysis in Section 9.1 under bounded per-resource contention. Table 9 shows F1 stabilizes for $\delta \geq 150$ ms and degrades sharply below 100 ms, indicating the paper's choice of $\delta = 150$ ms sits past the knee of the curve rather than being arbitrary.

Table 8. Empirical wall-clock scaling under bounded per-resource contention

Log Size (Events)	FRAG Time (s)	Baseline Time (s)
1,000	0.0017	0.0002
5,000	0.0047	0.0009
10,000	0.0128	0.0024
25,000	0.0424	0.0089
50,000	0.0867	0.0234
100,000	0.1949	0.0570
250,000	0.5104	0.1354

Note: FRAG = FHIR Resource Access Graph.

Table 9. F1 stabilization for $\delta \geq 50$ ms

δ (ms)	Precision	Recall	F1
50	0.892	0.724	0.799
100	0.906	0.857	0.881
150	0.917	1.000	0.957
200	0.916	1.000	0.956
300	0.914	1.000	0.955

Runtime scaled from 1.7 ms at 1,000 events to 510 ms at 250,000 events, near-linear in n, and well within the latency budget of both deployment modes in Section 8 (a full 250,000-event log, roughly a busy hospital-wide day of FHIR traffic, processes in half a second on a single core).

Sensitivity to the concurrency window δ:

Sensitivity to READ: WRITE workload ratio: Detection performance is essentially flat across a wide range of READ:WRITE ratios, indicating the model is not sensitive to the specific 3:1 ratio assumed in Section 5.3, shown in Table 10.

Sensitivity to timestamp resolution: F1 collapses at coarser timestamp resolutions, since rounding destroys the ordering information the TAV and CUR passes depend on. This gives the limitation noted in Section 10.2 (100 ms log granularity) a quantified, rather than asserted, basis, which is organized in Table 11.

Table 10. Detection across a wide range of READ:WRITE

Ratio (R:W)	Precision	Recall	F1
1:1	0.916	1.000	0.956
2:1	0.913	1.000	0.954
3:1	0.923	1.000	0.960
5:1	0.940	1.000	0.969
8:1	0.949	1.000	0.974

Table 11. Resolution with precision, recall, and F1 values

Resolution (ms)	Precision	Recall	F1
1	0.929	1.000	0.963
10	0.928	0.992	0.959
100 (paper's assumed resolution)	0.662	0.467	0.548
1000	0.520	0.369	0.432

Note: TAV = Time-of-Check-to-Time-of-Use Authorization Violation; CUR = Cascading Update Race.

A finding worth highlighting for the baseline-fairness question (Section 5.6): the same sensitivity code used above also confirms that a process-aware baseline variant's performance is contention-dependent, unlike FRAG's, whose three-pass structural matching is largely invariant to how densely accesses concentrate on a resource (subject to the timestamp-resolution effect above, which affects both detectors equally).

10. LIMITATIONS

This section is restructured in this revision to separate limitations of the FRAG model itself from limitations of the evaluation dataset and deployment context.

10.1 Model limitations

The three race classes defined here are not exhaustive. A fourth class, partial-synchronization races where not all clients implement ETag, is a natural extension. A fifth class, partial-order violations in distributed multi-server FHIR gateway architectures, is left for future work (Section 11), where the vector-clock literature discussed in Section 3.4 becomes directly relevant.

A single-resource claim architecture is employed: a resource is not re-checked once allocated to a race class in a detection cycle. This prevents race-class contamination (Section 5.5) but causes recall loss when a resource belongs to multiple race instances of different classes, quantified in Section 6.3/6.4. This trade-off favors precision over recall, which is acceptable for a safety-alert scenario but should be revisited with per-write claim tracking (Section 11).

As clarified in Section 3.5, FRAG's core detection scope (SWC, CUR) is non-adversarial by design; it detects races between correctly functioning, uncoordinated clients, not attacks by a malicious client capable of message reordering, forgery, or log tampering. Extending FRAG to that stronger threat model is future work, not a gap in the present model's stated scope.

The clinical-risk descriptions in Table 12 and Section 1 (e.g., "contraindicated drug") are illustrative motivating scenarios for why a given graph pattern matters, not a claim the simulation tests. FRAG's structural claim that a given access pattern constitutes a race, per Definitions 2–4, is what Section 6 validates; whether a specific instance of that pattern would actually change a clinical decision is a separate, clinical-semantics question outside FRAG's scope, and is the object of the clinical-partner validation proposed in Section 11.

Table 12. Healthcare race condition classification in FHIR Resource Access Graph (FRAG)

Class	FRAG Pattern	Illustrative Clinical Risk	Prop.
SWC	WRITE // WRITE, S = ∅	Lost prescription update	1
TAV	READ→act, WRITE intervenes	Authorization bypass	2
CUR	Stale READ, path depth ≥ 2	Contraindicated drug	3

Note: SWC = Simultaneous Write Conflict; TAV = Time-of-Check-to-Time-of-Use Authorization Violation; CUR = Cascading Update Race; FHIR = Fast Healthcare Interoperability Resources.
The Illustrative clinical risk column describes the motivating scenario from Section 1, not a claim tested by the simulation in Section 6.

10.2 Dataset and deployment limitations

The simulation uses artificial transaction logs. Usage-pattern assumptions for non-race event generation (3:1 READ:WRITE ratio; 2× frequency weighting for AllergyIntolerance and MedicationRequest) are based on general domain knowledge rather than measured clinical telemetry; Section 9.2 shows detection performance is not sensitive to the specific ratio chosen, which somewhat mitigates this concern, but actual usage patterns may still differ for high-write or high-frequency workloads not represented in the tested range.

The C3 partial-ETag-adoption model assigns protection independently per write event; Section 5.4 states explicitly that this does not reflect the vendor-clustered adoption pattern expected in practice, and proposes the clustered alternative as a near-term validation step (Section 11).

The FRAG model requires access logs to be available at the resolution and schema described in Section 5.2/8. Proprietary EHR vendors may not expose access logs natively, and internal logging infrastructure may not capture all concurrent access events handled before the FHIR layer; Section 8 proposes a proxy-based logging shim as a practical mitigation, though this only observes traffic passing through the proxy.
Timestamp resolution of 100 ms was used as a conservative estimate of FHIR server log granularity. Section 9.2 now quantifies this precisely: F1 falls from 0.96 at 10 ms resolution to 0.55 at 100 ms and 0.43 at 1000 ms, so in environments with coarser log granularity than 100 ms, detection reliability for FRAG (and, comparably, for the baseline) degrades substantially, regardless of the model's formal properties.

11. FUTURE WORK

The nearest-term validation step, requiring no new data access, is testing the vendor-clustered ETag adoption model proposed in Section 5.4 against the existing synthetic pipeline, to check whether C3's results are sensitive to adoption clustering rather than independent per-write randomization.

The next step is real-data validation with a clinical partner, involving institutional review board (IRB) approval to access FHIR server logs and a strategy to reverse-engineer approximate ground truth from log evidence; for instance,

using FHIR resource version conflicts (a conditional PUT failing on an ETag mismatch) as a proxy for observable SWC races. This is also the step that would test the clinical-severity claims noted as illustrative in Section 10.1, as distinct from FRAG's structural claims already validated in Section 6.

The mitigation counterpart to this detection framework is synchronization primitives for FHIR APIs, a second research direction, a non-trivial design challenge given HTTP's stateless nature. Candidate solutions include mandatory server-side ETag enforcement, a session-based resource reservation protocol, and CRDT-based conflict-free resource types for selected FHIR resources (e.g., Observation).

Extension to distributed multi-server FHIR architectures is a third direction. Modern cloud-native healthcare platforms deploy FHIR functionality across microservices with independent databases and clocks. Extending FRAG to this setting would require a vector-clock or partial-order timestamp model (Section 3.4 studies [28, 29]), well established in distributed systems but not yet applied to FHIR.

Finally, per-write claim tracking (Section 4.2.1, 6.4, and 10.1) is proposed to recover co-occurring races of different classes on the same resource without reintroducing race-class contamination, and extending the threat model of Section 3.5 to cover an adversary capable of log tampering or reordering is proposed as the natural security-hardening direction beyond the present non-adversarial scope.

12. CONCLUSIONS

This paper formally states a problem that, to the best of our knowledge and the search protocol in Section 2.4, had not been formally stated before: what does a race condition look like in a healthcare interoperability system, and how would one detect it?

The critical-section problem is shown to be formally related to FHIR-based healthcare data exchange. FRAG represents this mapping as a six-component directed graph in which race conditions are not timing accidents but structural properties. The three race-condition classes (SWC, TAV, CUR) carry structural correspondence lemmas relating them to FRAG's graph structure, soundness, and δ-completeness results relating Algorithm 1's detections to those formal definitions, with completeness holding conditional on an accurately specified synchronization set S, as discussed in Section 4.2.1.

The CUR class deserves special emphasis as the most critical and least intuitive failure mode: a patient can receive a contraindicated drug not because any system failed, but because each system performed correctly given its own input. The flaw is a property of access order, visible only at the graph level and invisible to any single system's audit trail, precisely the kind of problem formal methods are meant to help prevent.

Propositions 1–5 are mathematical results whose truth is independent of the simulation. The simulation demonstrates the detection approach's validity under controlled, synthetic evaluation conditions, pending the real-deployment validation proposed in Section 11: FRAG reaches a 90.0% F1 under full concurrent conditions, a 64.5 percentage-point improvement over the time-window baseline, with zero false positives on the sequential control condition. This study provides a documented search protocol substantiating the novelty claim, an explicit non-adversarial system and threat model, a procedure for constructing S in practice, a deployment architecture spanning batch and streaming modes with an explicit answer to the log-availability constraint, and a theoretical and empirical computational-complexity and sensitivity analysis showing near-linear scaling to 250,000 events and identifying timestamp resolution as the dominant sensitivity factor, together clarifying the approach's feasibility and tractability.

ACKNOWLEDGMENT

The authors recognize the foundational work of Dijkstra's 1965 critical-section formalism, which motivated the connection between traditional OS theory and modern healthcare infrastructure. The research was not supported by external funding.

REFERENCES

[1] HL7 International. (2019). HL7 FHIR Release 4 Specification. https://hl7.org/fhir/R4/.

[2] Dijkstra, E.W. (1965). Solution of a problem in concurrent programming control. Communications of the ACM, 8(9): 569. https://doi.org/10.1145/365559.365617

[3] TXOne Networks. (2026). Health level 7 communication protocol: Vulnerabilities and mitigation. https://www.txone.com/blog/hl7-protocol-vulnerabilities-mitigation/.

[4] Upadhyay, A., Laxmi, V., Naval, S. (2023). Navigating the concurrency landscape: A survey of race condition vulnerability detectors. arXiv preprint arXiv:2312.14479. https://doi.org/10.48550/arXiv.2312.14479

[5] Li, T., Bai, J.J., Han, G.D., Hu, S.M. (2024). LR-Miner: Static race detection in OS kernels by mining locking rules. In Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24), Philadelphia, PA, USA, pp. 6149-6166. https://dl.acm.org/doi/10.5555/3698900.3699244.

[6] Feng, Z.C., Chen, Y.M., Zhang, K.W., Li, X.F., Liu, G.J. (2025). Concurrency bug detection via static analysis and large language models. Future Internet, 17(12): 578. https://doi.org/10.3390/fi17120578

[7] Yang, C.Y., Zhao, Z.J., Xie, Z.C., Li, H.Y., Zhang, L.M. (2025). Knighter: Transforming static analysis with LLM-synthesized checkers. In Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles, Seoul, Republic of Korea, pp. 655-669. https://doi.org/10.1145/3731569.3764827

[8] Gong, S.S., Peng, D.L., Altınbüken, D., Fonseca, P., Maniatis, P. (2023). Snowcat: Efficient kernel concurrency testing using a learned coverage predictor. In Proceedings of the 29th Symposium on Operating Systems Principles, Koblenz, Germany, pp. 35-51. https://doi.org/10.1145/3600006.3613148

[9] NIST National Vulnerability Database. (2024). CVE-2024-52007: HAPI FHIR XXE Injection. https://nvd.nist.gov/vuln/detail/CVE-2024-52007.

[10] Ferreira, J.C., Elvas, L.B., Correia, R., Mascarenhas, M. (2024). Enhancing EHR interoperability and security through distributed ledger technology: A review. Healthcare, 12(19): 1967. https://doi.org/10.3390/healthcare12191967

[11] Maguluri, D.S., Velagala, L.P., Hossain, G. (2024).

Securing Healthcare 5.0: Exploring BYOD cyber risks, misuse cases, and best practices. In 2024 12th International Symposium on Digital Forensics and Security (ISDFS), San Antonio, TX, USA, pp. 1-6. https://doi.org/10.1109/ISDFS60797.2024.10527340

[12] Velagala, L.P., Hossain, G. (2023). Analyzing insider threats and human factors in Healthcare 5.0. In 2023 IEEE 20th International Conference on Smart Communities: Improving Quality of Life Using AI, Robotics and IoT (HONET), Boca Raton, FL, USA, pp. 95-100. https://doi.org/10.1109/HONET59747.2023.10374733

[13] Ayaz, M., Pasha, M.F., Alzahrani, M.Y., Budiarto, R., Stiawan, D. (2021). The fast health interoperability resources (FHIR) standard: Systematic literature review of implementations, applications, challenges and opportunities. JMIR Medical Informatics, 9(7): e21929. https://doi.org/10.2196/21929

[14] Tabari, P., Costagliola, G., De Rosa, M., Boeker, M. (2024). State-of-the-art fast healthcare interoperability resources (FHIR)–based data model and structure implementations: Systematic scoping review. JMIR Medical Informatics, 12: e58445. https://doi.org/10.2196/58445

[15] Bae, Y.S., Park, Y., Lee, S.M., et al. (2022). Development of blockchain-based health information exchange platform using HL7 FHIR standards: Usability test. IEEE Access, 10: 79264-79271. https://doi.org/10.1109/ACCESS.2022.3194159

[16] Yang, C.N., Li, P., Cheng, H.H., Kuo, H.C., Lu, M.C., Xiong, L.Z. (2024). A security model of multihospital FHIR database authorization based on secret sharing and blockchain. IEEE Internet of Things Journal, 11(6): 10325-10335. https://doi.org/10.1109/JIOT.2023.3328989

[17] Yaqub, N., Zhang, J.B., Khalid, M.I., et al. (2025). Blockchain enabled policy-based access control mechanism to restrict unauthorized access to electronic health records. PeerJ Computer Science, 11: e2647. https://doi.org/10.7717/peerj-cs.2647

[18] Cobrado, U.N., Sharief, S., Regahal, N.G., Zepka, E., Mamauag, M., Velasco, L.C. (2024). Access control solutions in electronic health record systems: A systematic review. Informatics in Medicine Unlocked, 49: 101552. https://doi.org/10.1016/j.imu.2024.101552

[19] Raducu, R., Rodríguez, R.J., Álvarez, P. (2022). Defense and attack techniques against file-based TOCTOU vulnerabilities: A systematic review. IEEE Access, 10: 21742-21758. https://doi.org/10.1109/ACCESS.2022.3153064

[20] Zukaib, U., Cui, X.H., Hassan, M., Harris, S., Hadi, H.J., Zheng, C.L. (2023). Blockchain and machine learning in EHR security: A systematic review. IEEE Access, 11: 130230-130256. https://doi.org/10.1109/ACCESS.2023.3333229

[21] Mahadik, S.S., Pawar, P.M., Muthalagu, R., et al. (2024). Digital privacy in healthcare: State-of-the-art and future vision. IEEE Access, 12: 84273-84291. https://doi.org/10.1109/ACCESS.2024.3410035

[22] He, Y., Aliyu, A., Evans, M., Luo, C. (2021). Health care cybersecurity challenges and solutions under the climate of COVID-19: Scoping review. Journal of Medical Internet Research, 23(4): e21747. https://doi.org/10.2196/21747

[23] Garcia-Perez, A., Cigarro-Navarro, J.G., Sallos, M.P., Martinez-Caro, E., Chinnaswamy, A. (2023). Resilience in healthcare systems: Cyber security and digital transformation. Technovation, 121: 102583. https://doi.org/10.1016/j.technovation.2022.102583

[24] Li, E., Clarke, J., Ashrafian, H., Darzi, A., Neves, A.L. (2022). The impact of electronic health record interoperability on safety and quality of care in high-income countries: Systematic review. Journal of Medical Internet Research, 24(9): e38144. https://doi.org/10.2196/38144

[25] Gupta, R., Kanungo, P., Dagdee, N., Madhu, G., et al. (2023). Secured and privacy-preserving multi-authority access control system for cloud-based healthcare data sharing. Sensors, 23(5): 2617. https://doi.org/10.3390/s23052617

[26] Jiang, J.X., Ross, J.S., Bai, G. (2025). Ransomware attacks and data breaches in US health care systems. JAMA Network Open, 8(5): e2510180. https://doi.org/10.1001/jamanetworkopen.2025.10180

[27] Yankson, B., Barati, M., Bondzie, R., Madani, R. (2025). The rise of hacking in integrated EHR systems: A trend analysis of U.S. healthcare data breaches. Journal of Cybersecurity and Privacy, 5(3): 70. https://doi.org/10.3390/jcp5030070

[28] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7): 558-565. https://doi.org/10.1145/359545.359563

[29] Fidge, C.J. (1988). Timestamps in message-passing systems that preserve the partial ordering. Australian Computer Science Communications, 10(1): 56-66.