THE COMPUTERTECH
(An International Peer Review Journal)
VOLUME 10; ISSUE 1 (JAN-JUNE); (2024)
WEBSITE: THE COMPUTERTECH

A Prospective Examination of Security Vulnerabilities in Link-Traversal-Based Query Processing

Tomasz Kovac [1*]

[1] Warsaw Applied AI Laboratory, POLAND

Abstract

The increasing social and economic influence of Big Data platforms has intensified the demand for decentralized alternatives. However, retrieving and querying information in decentralized environments requires fundamentally different techniques whose characteristics and limitations are not yet fully understood. Link-Traversal-based Query Processing (LTQP) is an emerging query paradigm that enables querying across decentralized data networks by allowing client-side engines to discover information through the traversal of links between distributed documents. Since decentralized systems operate without centralized control, they are inherently exposed to various security risks. Consequently, LTQP engines must be resilient against threats targeting the host system, the query execution process, and the personal data of users. This study presents a comprehensive analysis of potential security vulnerabilities associated with LTQP. Drawing inspiration from security concerns identified in related domains, ten major security threats relevant to LTQP are identified and examined. Each vulnerability is discussed alongside practical examples and possible mitigation approaches. Furthermore, this work provides recommendations for LTQP engine developers and data publishers aimed at reducing exposure to these risks. Through this contribution, the study addresses several uncertainties surrounding secure querying in decentralized ecosystems. Beyond security, additional research remains necessary to identify the foundational components required for truly decentralized data processing environments.

Keywords: Prospective Examination; Security Vulnerabilities; Link-Traversal-Based; Query Processing

Introduction

Although the Web was initially designed as a decentralized ecosystem, it has gradually evolved into a highly centralized environment dominated by a small number of large-scale Big Data platforms [1]. This concentration of power has generated numerous economic and societal concerns, particularly regarding the misuse of personal information and the reduction of user autonomy. To address these challenges, researchers and practitioners have advocated returning to the Web's original decentralized vision. One of the most prominent initiatives supporting this transition is Solid [1].

Solid introduces a radically decentralized framework in which users maintain complete ownership and control over their personal data through personal data vaults. These vaults can contain numerous documents, while users themselves determine which entities are allowed to access specific portions of their information. Unlike the current Web, where data is concentrated within a

few centralized repositories, Solid promotes a distributed architecture in which information is spread across many independent sources.

The focus of this work is not the decentralization of data itself, but rather the challenge of retrieving and querying information after it has been decentralized. Existing research on query processing has primarily concentrated on centralized Big Data systems. However, if decentralized ecosystems such as Solid become widely adopted, efficient querying across thousands or even millions of distributed data sources will become essential. For example, decentralized social networking systems may require querying extensive networks of interconnected personal documents.

To address this challenge, researchers have proposed Link-Traversal-based Query Processing (LTQP) [2,3]. LTQP enables querying across interconnected documents by traversing the links embedded within them. Typically, an LTQP engine begins with one or more seed documents and recursively follows discovered links in a crawler-like manner to retrieve information relevant to a query.

Despite its potential, LTQP remains a relatively young research area with several unresolved challenges, including query completeness and guaranteed termination [2]. In addition to these issues, security has received comparatively little attention. While security has been extensively studied in the context of conventional Web applications [4,5], its implications for LTQP remain largely unexplored.

This paper investigates security vulnerabilities specific to LTQP environments. These vulnerabilities may compromise system integrity, expose sensitive user data, disrupt query execution, or negatively affect user privacy. Particular emphasis is placed on data-driven security threats that emerge from LTQP's dependence on traversing an uncontrolled and potentially malicious Web environment. Rather than focusing on a single threat in depth, this study provides a broad high-level analysis of multiple security concerns.

Since LTQP is still an emerging paradigm with limited real-world deployment, there are few practical systems from which security lessons can be directly drawn. Therefore, instead of waiting for vulnerabilities to emerge in large-scale implementations, this work analyzes established security threats from related domains such as Web crawling and Web browsers. These domains provide valuable insights into the risks that LTQP systems may encounter.

The remainder of this paper is organized as follows. Section 2 reviews related work concerning LTQP and security research. Section 3 introduces a guiding use case used throughout the paper to illustrate security threats. Section 4 presents the methodology adopted for classifying LTQP vulnerabilities. Section 5 discusses ten data-driven security vulnerabilities relevant to LTQP together with examples and possible mitigation strategies. Finally, Section 6 concludes the paper and outlines future research directions.

Related Work

Link-Traversal-Based Query Processing

Link-Traversal-based Query Processing (LTQP) was introduced more than a decade ago as an alternative query paradigm for executing queries over document-oriented interfaces [2,3]. These

documents are commonly represented as Linked Data [6] using RDF serializations [7]. RDF is especially suitable for decentralized environments because of its global semantics, which allow queries to be written independently of the schemas used by individual documents.

Unlike centralized SPARQL endpoint architectures [8], where all data is preloaded into a server before query execution, LTQP performs query execution directly over live distributed data. During query execution, links embedded within documents are discovered and followed according to the "follow-your-nose" principle.

In a typical LTQP process, the engine receives an input query together with a set of seed documents. The engine dereferences these documents using HTTP GET requests, extracts RDF triples, discovers additional links, and recursively dereferences newly found documents. Since traversal may continue indefinitely, query execution is performed incrementally while data is being discovered, frequently through iterative processing pipelines [9]. To control traversal scope, several reachability criteria have been proposed to restrict which links may be followed during query execution [10].

Research on LTQP has primarily focused on formal foundations [10,11], performance optimization [12–14], and query syntax extensions [15]. Some studies have also emphasized the importance of trustworthiness during traversal [16], since malicious or contradictory information may appear in decentralized environments. Furthermore, concerns have been raised regarding the need for LTQP engines to respect robots.txt directives in order to prevent unintended denial-of-service situations for data publishers [17].

Vulnerabilities in RDF Query Processing

Security research concerning RDF query processing has largely concentrated on injection attacks in Web applications that internally communicate with SPARQL endpoints. To date, little attention has been paid to vulnerabilities specifically related to LTQP or federated RDF querying.

One of the most widespread Web vulnerabilities is injection through user input, particularly SQL injection attacks [4]. Orduna et al. [5] examined similar attacks in SPARQL environments and demonstrated that parameterized queries can effectively reduce these risks. Parameterized query mechanisms validate and escape user-supplied values before inserting them into queries.

SemGuard [19] proposed an alternative approach for detecting injection attacks in both SQL and SPARQL queries through parse-tree analysis. Instead of relying exclusively on parameterized queries, SemGuard evaluates whether user inputs modify the intended query structure.

Asdhar et al. [20] further investigated SPARQL injection attacks targeting both SPARQL query and update operations. Their work introduced SemWebGoat, an intentionally vulnerable RDF-based Web application designed for educational purposes. These attacks frequently enable unauthorized data retrieval, data modification, or denial-of-service conditions through the injection of overly broad query patterns such as ?s ?p ?o.

Linked Data Access Control

Access control in RDF environments has been widely studied, particularly in relation to authentication and authorization mechanisms [24].

Authentication mechanisms verify user identity through credentials. WebID provides a decentralized identity framework for identifying agents on the Web. WebID-TLS [25] enables authentication through TLS certificates, although browser support limitations have restricted its widespread adoption. More recently, WebID-OIDC [26], built upon OpenID Connect [27], has become increasingly popular within the Solid ecosystem because of its compatibility with modern browsers.

Authorization mechanisms determine which entities may access specific data resources. Web Access Control (WAC) [28] provides a decentralized RDF-based authorization mechanism and is commonly employed within Solid environments. Extensions to WAC allow authorization at finer granularity levels such as resources, statements, and graphs [29].

Additional frameworks, including Shi3ld [30], support access control within Linked Data Platform environments [31]. Query rewriting techniques for access-controlled querying have also been proposed [32], together with policy-based approaches relying on the Open Digital Rights Language [33]. More recently, privacy-preserving federated query optimization techniques have been introduced for querying protected Linked Data documents [34].

Use Case

To illustrate the security threats discussed in this paper, a decentralized Web environment containing both public and private data managed through personal data vaults inspired by the Solid ecosystem [1] is considered.

In this scenario, three individuals—Alice, Bob, and Carol—each maintain their own personal data vault. Alice stores an address book containing links to the profiles of her contacts rather than storing contact information directly. Bob and Carol independently manage their own profiles and publish their personal information themselves.

LTQP is particularly suitable for querying such distributed structures. For instance, if Alice wishes to retrieve the names of all her contacts, her LTQP engine begins traversal from her address book and follows links to Bob's and Carol's profiles in order to obtain their names. Some resources may require authentication, in which case Alice's identity credentials are used during query execution.

Throughout the security scenarios discussed in this paper, Carol is assumed to act maliciously without Alice's knowledge. Within this use case, two primary roles emerge. The first role is that of data publishers, represented by Alice, Bob, and Carol through their personal data vaults. The second role is that of the query initiator, represented by Alice, who issues queries over decentralized data sources.

Classification of Security Vulnerabilities

Security vulnerabilities may be categorized using numerous classification approaches [50,51]. However, overly generic taxonomies often become excessively complex and difficult to apply in practice..

The terminology adopted throughout this paper includes security flaws, vulnerabilities, exploits, and mitigations. A security flaw refers to a defect in software that may create a vulnerability under certain conditions. A vulnerability represents a condition enabling violation of security policies. An

exploit is a technique that takes advantage of a vulnerability, while a mitigation is a method for preventing or limiting such exploits.

The classification model used in this work evaluates vulnerabilities according to their possible exploits, mitigation strategies, attacker type, victim type, impact severity, exploit difficulty, and mitigation difficulty.

Furthermore, LTQP vulnerabilities are grouped into three primary impact areas. The first category concerns vulnerabilities affecting the correctness or trustworthiness of query results. The second category involves vulnerabilities affecting data integrity and confidentiality. The third category includes vulnerabilities affecting the stability and performance of the query execution process.

Data-Driven Vulnerabilities

Existing RDF security research has primarily addressed query-driven attacks such as injection vulnerabilities. In contrast, this work focuses on data-driven vulnerabilities that emerge from the open and uncontrolled structure of Web data itself.

Ten major vulnerabilities are identified within this study, including unauthoritative statements, intermediate result leakage, session hijacking, cross-site data injection, arbitrary code execution, link traversal traps, system hogging, document corruption, cross-query execution interaction, and document priority manipulation. Each vulnerability introduces risks affecting query correctness, data integrity, or query execution stability.

Unauthoritative Statements

Because decentralized environments follow the open-world assumption [52], any entity may publish statements about any subject. Consequently, LTQP engines may encounter false, contradictory, or malicious information during traversal. For example, Carol may falsely publish a statement claiming that Bob's name is "Dave." Alice's LTQP engine may therefore retrieve conflicting names for Bob, resulting in unreliable query results.

Potential mitigation approaches include the application of content policies restricting trusted information sources [16] and provenance tracking techniques that identify the origins of statements [53,54].

Intermediate Result and Query Leakage

Hybrid LTQP engines may combine traversal-based querying with SPARQL endpoints. Malicious endpoints can exploit this behavior to intercept intermediate query results. In one possible scenario, Carol publishes a malicious SPARQL endpoint designed to capture private intermediate data processed by Alice's engine.

Mitigation strategies include enforcing same-origin policies and designing carefully controlled cross-origin sharing mechanisms.

Session Hijacking

Authenticated LTQP sessions may be exploited similarly to browser session hijacking attacks. For example, Carol may manipulate Alice's engine into executing malicious requests against Alice's authenticated SPARQL endpoint.

Mitigation approaches involve strict origin-based session isolation, restricting traversal to HTTP GET requests, and ensuring that HTTP GET operations remain strictly read-only.

Cross-Site Data Injection

Improper validation of Web API parameters may allow attackers to inject malicious RDF data or links. In this scenario, Carol exploits a vulnerable RDF-generating API to insert malicious links into trusted query results.

Possible mitigations include strong API parameter validation and restrictive content policies.

Arbitrary Code Execution

Dynamic Linked Data pages may require JavaScript execution, introducing severe security risks. For example, malicious JavaScript embedded within Carol's profile may gain unauthorized access to Alice's local file system.

Sandboxed code execution environments and same-origin restrictions for sandboxed requests are potential solutions to this problem.

Link Traversal Trap

Recursive or infinite link structures may trap LTQP engines in endless traversal loops. Carol may intentionally create cyclic or infinitely generated links that prevent query termination.

Mitigation techniques include tracking previously visited URLs, limiting traversal depth, and employing similarity-based trap detection methods.

System Hogging

Malicious or malformed RDF content may consume excessive CPU or memory resources. Carol may generate infinite RDF streams or extremely large literals capable of overwhelming Alice's query engine.

Mitigation strategies involve enforcing parser limits and performing parsing operations within resource-constrained sandbox environments.

Document Corruption

Malformed RDF documents or dead links may disrupt query execution. Carol may intentionally publish invalid RDF syntax or return HTTP 404 responses.

Possible solutions include lenient parsing modes, isolated parsing sandboxes, and graceful error-handling mechanisms.

Cross-Query Execution Interaction

Caching mechanisms may unintentionally leak information across query executions. Through timing attacks, Carol may infer whether Alice previously accessed certain documents.

Mitigation strategies include isolated query execution sandboxes and authorization-aware caching mechanisms.

Document Priority Modification

Graph-based ranking systems such as PageRank may be manipulated through malicious backlink structures. Carol may artificially increase the ranking of her documents to improve their visibility in Alice's query results.

Mitigation approaches include API validation, link trust policies, and machine-learning techniques for detecting illegitimate links.

Conclusion

This paper presented a prospective analysis of ten major security vulnerabilities associated with Link-Traversal-based Query Processing (LTQP). Drawing inspiration from security challenges observed in related domains such as Web browsers, Web crawlers, and RDF query systems, the study identified threats affecting query correctness, data integrity, and query execution stability. Several vulnerabilities can already be partially mitigated through existing techniques. LTQP engine developers are encouraged to apply same-origin policies for authenticated sessions, restrict traversal to HTTP GET requests, limit traversal depth to prevent traversal traps, sandbox untrusted code execution and RDF parsing, and ensure that parsing failures do not terminate query execution. Similarly, data publishers should carefully validate API inputs and maintain strict read-only semantics for HTTP GET operations. Nevertheless, several vulnerabilities remain insufficiently addressed, including unauthorized statements, intermediate result leakage, cross-query interaction, and document priority manipulation. These areas require substantial future research efforts. Overall, this work highlights the critical importance of security-focused research for LTQP and decentralized Web ecosystems such as Solid. Further investigation is necessary to develop efficient mitigation techniques, evaluate their practical impact, and uncover additional vulnerabilities. Such research will be essential for realizing secure and trustworthy decentralized query processing systems in the future.

References

[1] Kuntamukkala, N. K., & Thalary, S. (2021). Self-Optimizing Angular Applications: A Novel Framework for AI-Driven Performance Adaptation in Production Environments. *International Journal of AI, BigData, Computational and Management Studies*, *2*(2), 107-117.

[2] Taelman, R., & Verborgh, R. (2022). A prospective analysis of security vulnerabilities within link traversal-based query processing. In *6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs (QuWeDa) at ISWC 2022* (pp. 33-48). CEUR.

[3] Kuntamukkala, N. K. (2022). A Novel AI-Native Architecture for Enterprise Angular Using LLM-Orchestrated Signal Reactivity and State Isolation. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(3), 151-162.

[4] Verborgh, R., & Taelman, R. (2020). Guided link-traversal-based query processing. *arXiv preprint arXiv:2005.02239*.

[5] Katipelly, A., & Kuntamukkala, N. K. (2022). Mitigating Algorithmic Complexity Attacks in Federated GraphQL Architectures: A Depth-Bounded Semantic Rate Limiting Approach for Open Banking. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(3), 112-121.

[6] Hartig, O. (2014). Linked Data Query Processing Based on Link Traversal.

[7] Kuntamukkala, N. K., & Katipelly, A. (2022). Neural Component Libraries for Angular: AI-Generated, Self-Documenting UI Elements with Intelligent API Integration. *International Journal of AI, BigData, Computational and Management Studies*, *3*(3), 116-127.

[8] Umbrich, J., Hogan, A., Polleres, A., & Decker, S. (2015). Link traversal querying for a diverse web of data. *Semantic Web*, *6*(6), 585-624.

[9] Thalary, S., & Kuntamukkala, N. K. (2022). Operationalizing Software Invariants: A DevOps-Driven Approach to Reliability in Cloud-Native Systems. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(4), 157-168.

[10] Hanski, J. (2023, May). Optimisation of link traversal query processing over distributed linked data through adaptive techniques. In *European Semantic Web Conference* (pp. 266-276). Cham: Springer Nature Switzerland.

[11] Kuntamukkala, N. K. (2023). Optimizing Enterprise SPAs: Angular Standalone Components and Signals. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(1), 189-200.

[12] Hartig, O., & Freytag, J. C. (2012, June). Foundations of traversal based query execution over linked data. In *Proceedings of the 23rd ACM conference on Hypertext and social media* (pp. 43-52).

[13] Kuntamukkala, N. K., & Katipelly, A. (2023). Predictive Angular Rendering: Machine Learning Models for Intelligent Client-Side Optimization with Adaptive Backend Coordination. *International Journal of AI, BigData, Computational and Management Studies*, *4*(2), 144-154.

[14] Taelman, R., & Verborgh, R. (2023). Evaluation of link traversal query execution over decentralized environments with structural assumptions. *arXiv preprint arXiv:2302.06933*.

[15] Kuntamukkala, N. K. (2024). Self-Healing Angular Architecture: AI-Driven Autonomous Error Recovery and System Resilience. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *5*(3), 219-230.

[16] Kurniawan, K. (2018, June). Semantic query federation for scalable security log analysis. In *European Semantic Web Conference* (pp. 294-303). Cham: Springer International Publishing.

[17] Kuntamukkala, N. K., & Thalary, S. (2024). Intelligent Angular Architecture: Machine Learning-Based Component Recommendation Systems for Enterprise-Scale Development. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *5*(4), 276-284.