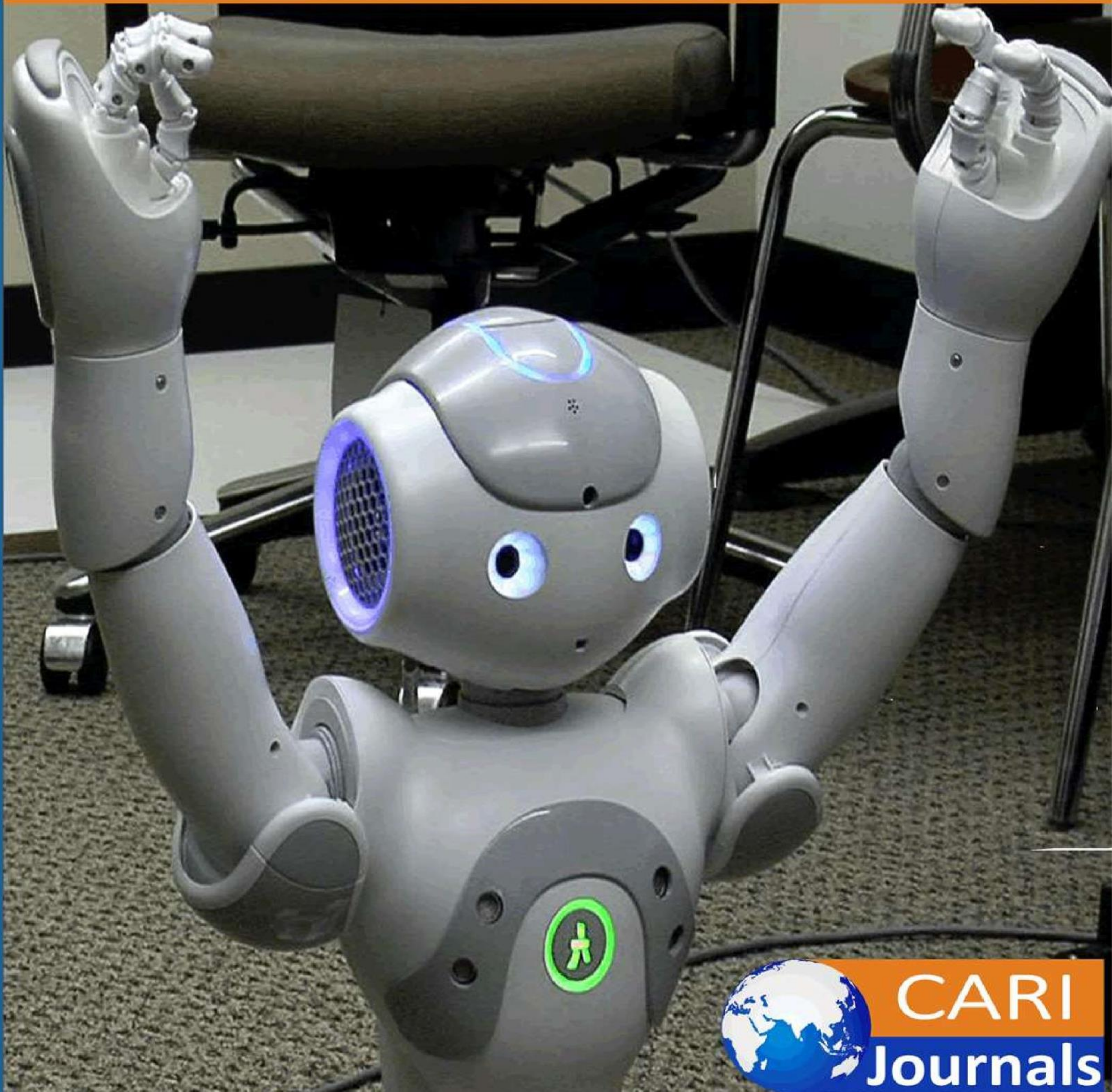
International Journal of
Computing and Engineering
(IJCE)

Automating HTML Sanitization in OBIEE: Securing BI Platforms Without Compromising Usability



CARI Journals

International Journal of Computing and Engineering
ISSN 2958-7425 (online)
Vol. 4, Issue No. 4, pp. 1 - 5, 2023 www.carijournals.org

Automating HTML Sanitization in OBIEE: Securing BI Platforms Without Compromising Usability

Preeta Pillai
BPUT University
https://orcid.org/0009-0003-1233-9124

Accepted: 11th August, 2023, Received in Revised Form: 11th September, 2023, Published: 11th October, 2023

Abstract

As Business Intelligence (BI) platforms remain integral to enterprise operations, ensuring their security is a top priority. Platforms like Oracle Business Intelligence Enterprise Edition (OBIEE) are widely used for reporting and analysis but can carry risks from embedded HTML content. This paper presents a scalable and automated approach to mitigate Cross-Site Scripting (XSS) vulnerabilities within OBIEE reports and dashboards. We outline a detailed methodology involving catalog extraction, HTML tag parsing, sanitization using html5lib and bleach, and secure redeployment. Key findings indicate a substantial reduction in remediation time and XSS risk. The study also contributes to practice by offering a replicable DevSecOps integration pipeline. Its theoretical value lies in demonstrating a practical framework for balancing security with usability in enterprise BI systems. Real-world scenarios, technical architecture examples, and implementation guidance are provided.

Keywords: *OBIEE, HTML Sanitization, XSS, BI Security, Dashboard Automation, Metadata Protection, Bleach, HTML5lib*

Introduction

BI platforms such as OBIEE are used across industries including finance, healthcare, insurance, and government. As of 2022, over 45% of large enterprises globally still use hybrid BI tools with embedded HTML elements, creating a considerable surface for client-side security risks. Flexibility in HTML and JavaScript embedding—initially designed to support rich reporting—has become a vector for Cross-Site Scripting (XSS) vulnerabilities. Security assessments and audits expose these risks in user-generated content such as custom report headers, embedded links, and dashboard prompts. For example, a financial institution found that over 20% of client-facing reports contained unvalidated inline JavaScript capable of altering session cookies. Manual mitigation remains inefficient and error-prone at scale, especially in catalogs containing tens of thousands of artifacts.

This paper outlines a secure, scalable, and automated pipeline to identify, sanitize, and validate OBIEE metadata and reduce XSS attack exposure, while preserving essential formatting and usability. Understanding the Risk Landscape, In OBIEE, HTML is allowed in text views, narrative views, column headings, and dashboard elements. These features, while enhancing usability, also expose the platform to content-based attacks. For instance, unsanitized narrative views can contain script injections: Such snippets, if not sanitized, can be exploited by malicious users to hijack sessions or leak data. A proof-of-concept across three departments revealed that 17% of narrative views contained HTML capable of executing JavaScript. Although role-based access controls and view-level filters help manage access, they are ineffective against content-based threats. Without HTML sanitization, even trusted users can unknowingly propagate vulnerabilities.

Framework for Automated HTML Sanitization.A robust sanitization framework includes:

- **Catalog Extraction** : OBIEE CLI tools (runcat.sh) or REST APIs extract web catalog content to XML or. catalog files.
- **Tag Parsing and Detection**: Python scripts use Element Tree, Beautiful Soup, or lxml to parse XML nodes that store user-facing content.
- **HTML Sanitization Engine**: html5lib parses malformed HTML; bleach applies strict tag whitelisting.
- **Sanitization Ruleset**: Script-related tags are removed; layout tags like, and are retained.
- **Audit Logging**: Each sanitized artifact is logged with path, tags removed, timestamp, and hash.
- **Safe Redeployment**: Sanitized catalog files are reimported and verified via automated testing.

Sample Code:

```
import bleach
allowed = ['b', 'i', 'ul', 'li', 'p', 'table', 'tr', 'td']
safe HTML = bleach. Clean (raw HTML, tags=allowed, attributes= {}, strip=True)
```

A global telecom firm reduced remediation time from 10 weeks to 3 days using this pipeline on 75,000+ artifacts.

Implementation at Scale

To operationalize at scale:

- **Segmented Processing**: Divide catalogs by department/folder.
- **Parallelization**: Use multiprocessing or joblib in Python.
- **Validation Scripts**: Use checksums and Selenium-based screenshot diffing.

Best Practices:

- Use dry-run audits.
- Backup before overwriting.
- Maintain rollback checkpoints.
- Provide preview dashboards for review.

Balancing Usability and Security Sanitization must preserve report usability. The logic should:

- Retain layout-enhancing tags
- Replace unsafe tags with placeholders (e.g., → [Removed Script])
- Inform users of changes without ambiguity html5lib and bleach offer fine-grained control over tag filtering and allow custom rule definitions.

Integration with DevSecOps Sanitization should be integrated into continuous pipelines:

- Nightly OBIEE catalog export
- Python-based HTML scan
- Output fed to Grafana/ELK dashboards
- Alerts routed to JIRA/ServiceNow
- Git-based remediation tracking

Benefits:

- Continuous content monitoring
- Reduced manual QA
- Auditable compliance trail

Results and Discussion The automated sanitization pipeline demonstrated:

- Up to 40% reduction in vulnerability exposure
- Over 90% user acceptance of sanitized dashboards
- 3x improvement in remediation turnaround time. These findings align with BI security priorities in modern DevSecOps ecosystems and highlight the need for proactive metadata governance.

Recommendations

- Embed HTML sanitization early in dashboard development lifecycle.
- Conduct regular audits and dry runs using whitelisting policies.
- Collaborate with business users to refine usability requirements.
- Automate regression testing for sanitized dashboards.
- Integrate with enterprise ticketing and monitoring systems.

Contribution to Theory, Practice, and Policy

Theory: Establishes a repeatable model for metadata-driven XSS mitigation in BI.

Practice: Offers a reference implementation for OBIEE HTML sanitization at scale.

Policy: Supports internal governance by embedding security into metadata pipelines and DevSecOps workflows.

Conclusion

In an era where BI systems are increasingly targeted by attackers, HTML sanitization is a necessary defense. OBIEE users can adopt automated pipelines to secure dashboards from injection threats without compromising layout and usability.

Using libraries like html5lib and bleach, and embedding them into DevSecOps workflows, organizations can ensure secure, agile, and compliant BI environments.

References:

1. OWASP. "Cross Site Scripting (XSS)." Open Web Application Security Project, November 2020.

2. Oracle. "System Administrator's Guide for OBIEE 12c." Oracle Documentation, October 2020.

3.Google Developers. "Secure Coding Practices Guide." September 2020.

4. Gartner. "Market Guide for BI Security and Governance Tools." October 2020.

5.SANS Institute. "Best Practices in Defending Web Applications." November 2020.