Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Build-Aware Security Scanning for Modern JavaScript Applications: The SecureBuild Framework

Laxmi P Sarva

Independent Researcher

Email laxmipsarva@gmail.com

USA

ARTICLE INFO

Received: 01 Apr 2026

Revised: 18 May 2026

Accepted:28 May 2026

ABSTRACT

SecureBuild is a security scanning platform that is open source, enabling developers to quickly analyze their modern react and node.js applications to solve the critical security problems of today. Modern web apps rely on elaborate front-end build pipelines, distributed back-end services, and API-dependent architectures, forcing traditional security tools to be outdated in order to be able to uncover vulnerabilities deeply embedded within the application and fully operational misconfigurations. SecureBuild solves these problems by incorporating build-to-source mapping, a static taint-aware analysis, and a light-weight runtime validation into one security analysis framework. The platform has the ability to detect vulnerabilities like Cross-Origin Resource Sharing (CORS) Misconfigurations, Weak Content Security Policies (CSP), SQL Injection Vectors, Unsafe JavaScript Execution Patterns, and Suspicious Automated Bot Activities. SecureBuild uses source mapping to re-map minified frontend bundles to their original source code so that developers can easily trace vulnerabilities and speed up remediation efforts. Moreover, the lightweight runtime validation engine helps in detecting violations with ephemeral runtime checks, HTTP header validation, preflight request analysis, and browser-based CSP violation simulations using automation tools like Playwright and Puppeteer. SecureBuild is optimized for DevSecOps and CI/CD environments to deliver pull-request ready scans with low false-positive rates and confidence-based findings, remediation guidance, SARIF compliant reporting and seamless integration with platforms like GitHub Actions and GitLab CI. In addition to vulnerability detection, the platform also focuses on developer adoption with extensible rule systems and customizable configurations, suppression baselines, and streamlined onboarding to minimize integration friction in existing software development workflows. SecureBuild brings together the capabilities of traditional static analysis tools and the more comprehensive dynamic scanning solutions in a scalable, accurate and workflow-friendly way. The platform allows organizations to detect and fix vulnerabilities before they go into production, reduce production security risks, create developer trust in security tools, and enhance the security of modern web infrastructures.

Keywords: SecureBuild, Web Application Security, DevSecOps, Static Analysis, Runtime Validation, React Security, Node.js Security, CI/CD Securit.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Introduction

With increased complexity of modern web applications, which use sophisticated Frontend frameworks and distributed Backend services, the attack surface is expanding and it is becoming more difficult to address various security issues like CORS mis-configuration, weak Content Security Policies (CSP) and SQL injection. In React and Node.js environments, these risks are often hidden from view because production deployments are likely to rely on minified front-end bundles, dynamically-generated server code, and dynamic CI/CD pipelines. Most traditional security tools isolate static code analysis, dependency scanning and runtime testing, leading to disjointed security visibility and remediation processes.

SecureBuild tackles this challenge with a developer-focused security scanning framework developed specifically for today's React and Node context. SecureBuild's build-to-source mapping, taint-aware static analysis and lightweight runtime validation will help find security flaws quickly, directly within the build and pull request workflow. With a focus on actionable remediation, low false-positive rates and seamless CI/IDE integration, the platform enables development teams to identify and address vulnerabilities earlier in the software delivery lifecycle without compromising deployment speed or operational efficiency.

Problem Statement

Today, complex build pipelines, third-party integrations and distributed back-end services are commonplace to build modern React or Node.js applications, leading to larger attack surfaces that are hard to monitor and secure properly. Production environments typically bundle up the front end code to be minified and dynamically generate the server code, which often makes it harder to spot vulnerabilities like SQL injection, insecure CORS policies and weak CSPs.

The current security infrastructure is usually based on static source code, dependency checking, or dynamic testing at the extreme end, leading to a lack of co-ordination, high false-positive ratios and a lack of knowledge of actual runtime behaviour. Many solutions also do not have efficient mapping between build and source, and are not developer-friendly for being integrated into modern CI/CD pipelines, resulting in slow remediation and high operational costs.

As organizations accelerate software delivery cycles, developers require lightweight, accurate, and actionable security validation that can identify vulnerabilities early within pull request and deployment workflows without disrupting productivity. This gap highlights the need for an integrated security scanning framework capable of combining source mapping, taint-aware analysis, and runtime validation specifically optimized for modern React and Node.js ecosystems.

SecureBuild Solution Overview

SecureBuild is an open-source-first application security scanning platform designed to identify critical security weaknesses in modern React and Node.js applications. Unlike traditional tools that focus only on source-code inspection or heavyweight runtime testing, SecureBuild combines build-to-source mapping, static taint-aware analysis, and lightweight runtime validation into a unified developer-centric framework.

The platform is specifically optimized for CI/CD and pull-request workflows, enabling development teams to detect and remediate vulnerabilities early without introducing excessive scan latency or operational complexity. By integrating directly into development pipelines, SecureBuild provides fast, actionable, and low-noise security feedback that improves both software security and developer productivity.

SecureBuild primarily targets three high-impact vulnerability classes commonly observed in modern web applications:

- Cross-Origin Resource Sharing (CORS) misconfigurations

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

- Content Security Policy (CSP) weaknesses
- SQL Injection (SQLi) vulnerabilities

The framework operates through three coordinated scanning engines:

1. **Build-to-Source Mapping Engine**

   Analyzes production frontend bundles and leverages source maps to trace detected issues back to original source files and code locations.

2. **Static Taint-Aware Analysis Engine**
   Uses Abstract Syntax Tree (AST) parsing and lightweight data-flow tracking to identify unsafe handling of untrusted inputs across Node.js and frontend components.

3. **Runtime Validation Engine**
   Executes ephemeral runtime checks to validate real HTTP security headers, simulate CORS preflight requests, and detect runtime CSP violations using browser automation tools.

Table: Core Solution Components

Component	Function	Security Benefit
Build-to-Source Mapping	Maps vulnerabilities in minified frontend bundles back to original source code	Improves remediation accuracy and developer traceability
Static Taint Analysis	Tracks untrusted inputs flowing into sensitive sinks such as database queries	Detects SQL injection and unsafe coding patterns
Runtime Header Validation	Validates live CSP and CORS configurations during execution	Detects deployment-time misconfigurations
Browser Runtime Checks	Uses Playwright/Puppeteer for frontend security validation	Identifies inline scripts, eval usage, and CSP violations
CI/CD Integration	Integrates with GitHub Actions, GitLab CI, and IDE workflows	Enables continuous security testing during development
Developer-Focused Reporting	Generates SARIF, JSON, and HTML reports with remediation guidance	Reduces false positives and accelerates issue resolution

A key differentiator of SecureBuild is its emphasis on developer experience (DX). Rather than functioning as a heavy penetration-testing platform, the system is engineered for rapid feedback cycles, low operational

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

overhead, and actionable remediation. This approach allows organizations to shift security validation earlier into the software development lifecycle while maintaining development velocity.

Core Features and Capabilities

SecureBuild combines static analysis, build artifact inspection, and lightweight runtime validation into a unified developer-focused security scanning platform for modern React and Node.js applications. Its architecture is designed to provide fast, actionable, and low-noise security feedback directly within CI/CD workflows.

Frontend Build Security Scanning

SecureBuild analyzes production-ready frontend bundles and uses source-map correlation to trace vulnerabilities back to their original source files and line numbers. This capability enables developers to identify risks embedded within minified JavaScript builds without manually reverse-engineering compiled assets.

The scanner detects:

- Unsafe eval() and new Function() usage
- Inline script execution
- dangerouslySetInnerHTML misuse
- Suspicious DOM parsing operations
- CSP violations in production builds

Static Taint-Aware Analysis

The platform implements lightweight taint-flow analysis to track untrusted user inputs from sources to dangerous execution sinks. This approach improves SQL injection detection accuracy while minimizing false positives commonly associated with rule-only scanners.

Supported ecosystems include:

- PostgreSQL (pg)
- MySQL
- Sequelize
- Prisma
- Raw SQL execution patterns

SecureBuild identifies:

- Query string concatenation
- Unsafe template literals
- Non-parameterized database queries
- Dangerous ORM raw query methods

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Runtime Security Validation

Unlike purely static scanners, SecureBuild performs controlled runtime validation against ephemeral application instances. This enables verification of real security behavior rather than relying solely on source inspection.

Runtime checks include:

- CORS preflight simulation
- CSP header verification
- Detection of permissive cross-origin configurations
- Browser-based validation using Playwright/Puppeteer
- Runtime evaluation and inline script monitoring

Build-Time Bot Detection

SecureBuild incorporates lightweight behavioral inspection mechanisms to identify indicators of automated abuse and malicious bot activity during frontend and server runtime checks.

Detection capabilities include:

- Suspicious JavaScript execution patterns
- Known bot signatures
- Automated request behavior analysis
- Potential scraping and abuse indicators

This functionality is particularly valuable for small businesses seeking early detection of malicious automation before deployment.

Developer-Centric Reporting and CI Integration

SecureBuild is optimized for modern DevSecOps workflows and integrates directly into pull request pipelines and continuous integration environments.

Key developer-focused capabilities include:

- SARIF-compliant reporting
- GitHub Action integration
- JSON and HTML security reports
- Pull request annotations
- Confidence scoring and remediation guidance
- Rule suppression baselines for noise reduction

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Table 2. SecureBuild Core Capability Summary

Capability Area	Primary Function	Key Technologies	Security Benefits
Frontend Build Scanning	Analyze production bundles and map findings to source	Source maps, AST parsing	Faster remediation and improved traceability
Static Taint Analysis	Track untrusted input flows to dangerous sinks	Lightweight taint engine	Accurate SQL injection detection with fewer false positives
Runtime Validation	Validate live application behavior	Playwright, Puppeteer, ephemeral runtimes	Detection of real-world CORS and CSP misconfigurations
Bot Detection	Identify automated abuse patterns	Behavioral analysis and signature matching	Reduced scraping and malicious automation risk
CI/CD Integration	Deliver security findings during development	SARIF, GitHub Actions, JSON reporting	Faster developer feedback and secure deployment workflows

Extensibility and Future Expansion

SecureBuild uses a modular plugin-based architecture that allows organizations to extend scanning rules and customize detection logic for project-specific requirements. Planned extensions include:

- Cross-site scripting (XSS) detection
- Server-side request forgery (SSRF) analysis
- Open redirect validation
- Compliance-focused rule packs aligned with OWASP and PCI standards

This extensible approach ensures that SecureBuild can evolve alongside emerging web application security threats and enterprise security requirements.

System Architecture

SecureBuild adopts a modular and developer-centric architecture designed to provide fast, accurate, and scalable security analysis for modern React and Node.js applications. The platform combines static code analysis, build-to-source mapping, and lightweight runtime validation into a unified security scanning workflow.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

At the core of the system is the SecureBuild CLI engine, developed using TypeScript and Node.js. The CLI orchestrates scanning operations, manages rule execution, processes reports, and integrates directly into local development environments and CI/CD pipelines. This lightweight design enables rapid execution during pull requests and automated deployment workflows.

The architecture consists of three primary analytical layers:

1. Build-to-Source Mapping Layer

This layer analyzes production frontend bundles and utilizes source maps to trace minified JavaScript code back to original source files and line numbers. By leveraging source-map consumers, SecureBuild can identify unsafe constructs such as inline scripts, eval, and dynamically generated functions while preserving developer visibility into the originating source code. This significantly improves remediation speed and reduces debugging complexity.

2. Static Taint-Aware Analysis Engine

The static analysis subsystem uses Abstract Syntax Tree (AST) parsing through tools such as @babel/parser and the TypeScript Compiler API. SecureBuild implements lightweight taint-flow tracking to monitor the propagation of untrusted user input toward sensitive execution sinks, including database queries and dynamic code execution points. The engine supports common Node.js database drivers and ORMs such as PostgreSQL, MySQL, Sequelize, and Prisma, enabling effective detection of SQL injection vulnerabilities and insecure raw query usage.

3. Runtime Validation Environment

To complement static analysis, SecureBuild performs ephemeral runtime validation using isolated local or containerized execution environments. Automated browser frameworks such as Playwright and Puppeteer simulate real frontend interactions to validate Content Security Policy (CSP) enforcement, detect inline script execution, and identify runtime eval behavior. Simultaneously, the runtime engine evaluates HTTP response headers and performs simulated CORS preflight requests to uncover insecure cross-origin configurations.

The reporting subsystem generates SARIF-compliant outputs, JSON reports, and HTML summaries to support IDE integration, CI dashboards, and automated pull request annotations. SecureBuild also provides extensibility through a plugin-based rule framework, enabling organizations to customize policies, add new vulnerability checks, and manage suppression baselines according to project-specific security requirements.

Overall, the SecureBuild architecture emphasizes scalability, low false-positive rates, and seamless developer experience, making it well-suited for modern DevSecOps environments and continuous security validation workflows.

Threat Model and Security Scope

SecureBuild is designed to identify common and high-impact security risks in modern React and Node.js applications, with a primary focus on CORS misconfigurations, weak Content Security Policies (CSP), and SQL injection vulnerabilities. The platform operates under a developer-centric threat model that prioritizes early detection during software development and CI/CD workflows.

The threat model assumes that attackers may exploit insecure frontend scripts, permissive cross-origin policies, unsafe database query construction, or weak runtime security headers to gain unauthorized access,

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

inject malicious code, or exfiltrate sensitive information. SecureBuild addresses these threats through a combination of static taint-aware analysis, build-to-source mapping, and lightweight runtime validation.

The system specifically targets:

- SQL injection vectors caused by unsanitized user input and unsafe ORM/raw query usage
- CSP weaknesses such as unsafe-inline, unsafe-eval, and missing script-src directives
- Insecure CORS configurations including wildcard origins with credential sharing
- Unsafe frontend behaviors such as eval, new Function, and dangerouslySetInnerHTML
- Automated bot activity and suspicious runtime behaviors during ephemeral execution checks

SecureBuild is intended for controlled development and testing environments and does not replace full penetration testing, advanced runtime monitoring, or dependency vulnerability management platforms. Runtime validation is performed only in isolated ephemeral environments to avoid risks to production systems.

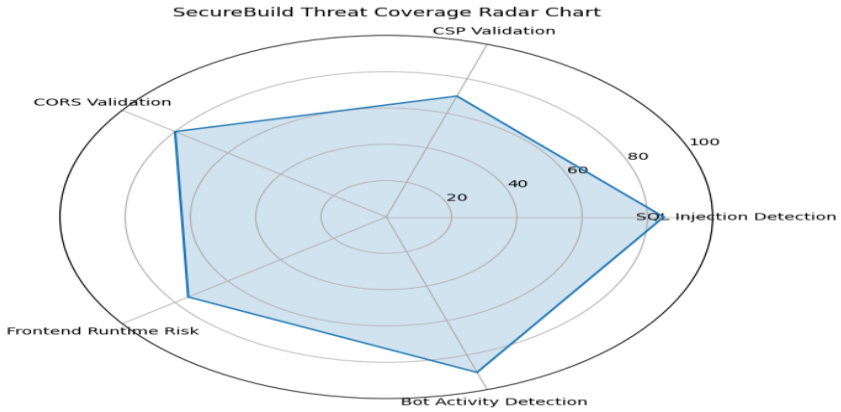

Fig: This radar chart illustrates an indicative security coverage profile for SecureBuild across five key threat detection and validation domains. Scores represent relative coverage strength on a normalized scale from 0 to 100, where higher values indicate stronger detection capability or validation depth. The visualization is intended for conceptual assessment and comparative analysis in a cybersecurity architecture context rather than a measured production audit result.

Business & Adoption Strategy: SecureBuild

SecureBuild's business and adoption strategy is grounded in a DevSecOps "shift-left" distribution model, where security tooling is embedded directly into developer workflows (CI/CD, PR checks, and IDE feedback loops). This aligns with modern industry adoption patterns where organizations increasingly prioritize automated security validation within GitHub-based pipelines and CI systems to reduce supply-chain and application-layer risk early in development cycles .

1. Open-Source Core Adoption Model

SecureBuild adopts a core open-source strategy (MIT/BSD license) to maximize developer trust, transparency, and rapid community-driven iteration. This lowers adoption friction and mirrors successful DevSecOps tooling ecosystems where open tooling becomes the de facto entry point for enterprise security

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

standardization. The OSS core functions as the “on-ramp” for individual developers and small teams, enabling organic virality through GitHub repositories, starter rulesets, and community contributions.

2. CI/CD-Native Distribution (Primary Channel)

Adoption is driven primarily through:

- GitHub Action marketplace integration
- GitLab CI templates
- Pre-commit hooks and CLI installs

This ensures SecureBuild is activated where code is written and merged, not as a separate security phase. This approach aligns with industry DevSecOps practice, where automation in CI pipelines is the dominant adoption vector for security tools .

3. Developer-First Growth Strategy

SecureBuild is designed for PR-native security feedback loops, meaning findings are delivered as:

- PR annotations
- SARIF outputs (IDE integration)
- Inline remediation suggestions

This “developer experience (DX) first” model increases adoption because teams are more likely to integrate tools that do not disrupt velocity or require specialist security expertise.

4. Enterprise Monetization Layer

While the core remains open-source, monetization is structured around:

- Hosted SaaS dashboards (cross-repo visibility and analytics)
- Enterprise rule packs (PCI-DSS, OWASP compliance baselines)
- On-premise deployment for regulated industries
- Advanced bot detection and runtime intelligence modules

This follows a common open-core DevSecOps commercialization pattern where enterprises pay for scale, governance, and compliance visibility, while developers use the free CLI and CI tools.

5. Community & Ecosystem Expansion

Growth is reinforced through:

- Security-focused GitHub repositories and example vulnerable apps
- Community rule contributions (plugin ecosystem)
- Integration partnerships with DevSecOps tooling (SAST, secrets scanning tools)
- Educational content (taint analysis, CSP/CORS hardening tutorials)

This mirrors observed DevSecOps ecosystem expansion patterns where adoption accelerates via shared tooling repositories and collaborative security frameworks .

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

6. Adoption Metrics & Feedback Loop

SecureBuild tracks:

- PR-level scan activation rates
- False positive suppression trends
- CI failure-to-fix time
- Rule effectiveness per vulnerability class (CORS, CSP, SQLi)

These metrics are used to continuously refine detection rules and reduce noise, reinforcing trust and long-term adoption in engineering teams.

Benefits and Impact of SecureBuild

SecureBuild delivers measurable improvements in software security posture, developer efficiency, and CI/CD reliability by integrating build-to-source mapping, taint-aware analysis, and lightweight runtime validation. Its impact is most significant in reducing late-stage vulnerability discovery and improving remediation speed within modern DevSecOps pipelines.

Key Benefits

- **Early Vulnerability Detection (Shift-Left Security):**

SecureBuild identifies CORS, CSP, and SQL injection issues during PR/CI stages, aligning with DevSecOps best practice of shifting security left to reduce remediation cost and deployment risk .

- **Reduced False Positives through Context-Aware Analysis:**

Taint-flow tracking and source-map resolution improve precision by linking runtime or build findings directly to original source code, reducing noise common in traditional SAST tools .

- **Faster Remediation Cycles (Lower MTTR):**

Developer-centric outputs (SARIF, PR annotations, and fix suggestions) shorten Mean Time to Repair by enabling direct fixes inside CI workflows rather than post-deployment debugging.

- **Improved CI/CD Throughput:**

Lightweight scans designed for PR-time execution prevent pipeline bottlenecks while maintaining continuous security validation, supporting high-frequency deployment environments .

Table: Impact

Impact Area	SecureBuild Contribution	Observed Effect in DevSecOps Literature
Vulnerability Detection Time	PR/CI-stage detection (shift-left)	Earlier detection reduces security debt
Remediation Efficiency	Direct source-mapped fixes + PR annotations	Lower MTTR and faster patch cycles

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

False Positive Reduction	Taint-aware + runtime validation correlation	Improved detection precision and trust
Pipeline Performance	Lightweight, non-blocking scans	Maintains CI/CD speed under security load

Implementation Challenges and Risk Mitigation

Despite its promise, SecureBuild does not come without its set of challenges in the real-world context of CI/CD environments, particularly in the context of complex JavaScript ecosystems and variable build pipelines.

One of the major difficulties is the source-map dependency and accuracy. One of the core features of SecureBuild is creating build-to-source mapping from source maps generated when building the front-end code. The availability of inconsistent or missing source maps during production can have a major impact on traceability of detected issues to source code. This is addressed by enforcing policies on CI that specify deterministic builds with embedded or stored source maps, and heuristics to try to approximate bundle-level locations when there is no source map.

One of the difficulties is runtime execution variability. When running in a lightweight runtime validation, Node servers and frontend builds are ephemeral and are potentially different in local, containerized, and CI environments. This adds inconsistencies to CSP/CORS validation and dynamic script detection. Containerized execution environments (e.g., Docker based runners), normalizing the execution environment, and simulated consistent request patterns via deterministic test harnesses are all mitigation strategies.

Another restriction is the production of false positives and noise, especially for taint analysis of TypeScript/JavaScript code. Data-flow tracking can become confusing in dynamic languages because of the features it offers, higher-order functions, and framework abstractions (such as the ORM layer or a dependency injection pattern). To overcome this, SecureBuild employs confidence scoring, rule prioritization and suppression baselines, which enable teams to be sensitive to the individual project without losing signal quality in CI workflows.

Another aspect that can cause overhead in CI pipelines with large monorepos or complex React builds is performance. Incremental scanning, caching of AST and dependency graphs, and PR-diff based analysis are the key parts in mitigation to restrict execution scope to modified code paths.

Lastly, there is an evolving threat surface, with increasing attack patterns and diversity of frameworks. Static rules and taint models may become outdated as new frameworks, ORMs and bundlers are created. This is addressed through a plugin-based architecture, enabling community-contributed detection rules and continuous updates aligned with OWASP guidance and emerging web security research trends.

Collectively, these mitigations ensure SecureBuild remains both practical for CI/CD integration and resilient against the inherent complexity of modern full-stack JavaScript security analysis.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Future Roadmap (SecureBuild)

SecureBuild's roadmap extends its current capabilities in build-aware scanning, taint analysis, and lightweight runtime validation toward a more autonomous, ecosystem-wide application security platform aligned with modern DevSecOps and supply-chain security trends.

1. Expanded Vulnerability Coverage (XSS, SSRF, Open Redirects)

Extend the current focus beyond CORS, CSP, and SQL injection to include higher-order application risks such as cross-site scripting (XSS), server-side request forgery (SSRF), and open redirect vulnerabilities. This aligns with industry movement toward broader semantic code coverage rather than point-rule detection.

2. AI-Augmented Taint Analysis and Code Reasoning

Introduce LLM-assisted static analysis to improve taint-flow precision, reduce false positives, and detect complex multi-step injection chains that traditional SAST tools struggle with. This follows the broader industry shift toward AI-assisted vulnerability triage and remediation in CI pipelines .

3. Federated and Continuous Supply Chain Security

Evolve SecureBuild into a federated scanning system capable of analyzing dependencies, SBOM graphs, and transitive package risks across CI/CD ecosystems. This reflects the growing importance of software supply chain security as the primary attack surface in modern applications .

4. Agentic CI/CD Security Enforcement

Introduce policy-driven, agent-like enforcement within pipelines that can automatically block risky builds, enforce remediation PRs, and validate security posture continuously across commits. This aligns with the emerging shift toward "shift-left + autonomous CI/CD security controls" in DevSecOps pipelines .

5. Real-Time Runtime Intelligence Layer

Enhance runtime validation with persistent, low-overhead monitoring that correlates build-time findings with live application behavior. This enables detection of misconfigurations that only manifest under production-like traffic patterns.

6. Compliance-Ready Security Posture (SOC2, NIS2, OWASP, CRA)

Integrate standardized reporting layers (SARIF+, audit trails, and policy-as-code outputs) to support enterprise compliance and regulatory frameworks, making SecureBuild suitable for regulated environments.

7. Plugin Ecosystem and Community Rule Marketplace

Develop a modular plugin architecture where security researchers and organizations can publish custom detection rules, industry-specific policies, and curated exploit signatures, accelerating community-driven expansion.

8. Self-Learning Detection System

Implement feedback loops from developer fixes and runtime outcomes to continuously refine rule accuracy, prioritize high-risk patterns, and adapt detection logic over time.

The roadmap transitions SecureBuild from a high-speed CI security scanner into a full-spectrum, intelligence-driven application security platform that unifies static analysis, runtime validation, and supply-chain awareness into a single developer-first system.

Journal of Information Systems Engineering and Management
2026, 11(3s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Conclusion

SecureBuild addresses a persistent gap in modern application security tooling by unifying build-to-source mapping, static taint-aware analysis, and lightweight runtime validation into a single developer-centric pipeline. Unlike traditional security solutions that operate in isolation or generate high-noise outputs, SecureBuild prioritizes actionable, context-aware findings directly within CI/CD workflows, enabling vulnerabilities in CORS, CSP, and SQL injection pathways to be identified and remediated early in the development lifecycle.

By bridging minified production artifacts with original source context and validating real runtime behavior, the framework improves detection accuracy while reducing false positives—two critical limitations in existing approaches. Its extensible architecture and open-source-first model further position it as a scalable foundation for modern DevSecOps practices.

Overall, SecureBuild contributes a practical shift toward integrated, low-friction security enforcement, reinforcing the principle that secure software delivery must be embedded directly into developer workflows rather than treated as a post-deployment concern.

References

[1] Ahuja, R. (2022). Application of static application security testing (SAST) in CI/CD pipelines for early detection of insecure coding practices through syntax tree analysis and custom rule sets. *International Journal on Recent and Innovation Trends in Computing and Communication, 10*(2), 80–87. https://ijritcc.org/index.php/ijritcc/article/view/11922

[2] Jovanović, N., Kruegel, C., & Kirda, E. (2010). Static analysis for detecting taint-style vulnerabilities in web applications. *Journal of Computer Security, 18*(5), 861–907. https://doi.org/10.3233/JCS-2009-0385

[3] Thome, J., Shar, L. K., & Briand, L. (2015). Security slicing for auditing XML, XPath, and SQL injection vulnerabilities. *Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 553–564. https://doi.org/10.1109/ISSRE.2015.7381847

[4] Yuan, Y., Lu, Y., Zhu, K., Huang, H., Yu, L., & Zhao, J. (2023). A static detection method for SQL injection vulnerability based on program transformation. *Applied Sciences, 13*(21), 11763. https://doi.org/10.3390/app132111763

[5] Takon, A. (2024). Data-Driven Threat Intelligence for Energy and Critical Asset Management. *International Journal of Technology, Management and Humanities*, *10*(04), 253-266.

[6] Williams, M. O. Numerical Modelling of Reactive Transport in Geothermal Reservoirs for Long-Term Performance Prediction. International Journal of Environmental Sciences, 9(1s), 2023.

[7] Barua, S. (2025). Biochar-Enhanced Filtration Media For Multi-Pollutant Industrial Runoff. Journal of Data Analysis and Critical Management, 1(04), 95-102.

[8] Takon, A. (2024). Data Science Approaches to Asset Integrity Management in Offshore and Onshore Oil and Gas Operations. *Multidisciplinary Innovations & Research Analysis*, *5*(2), 17-31.

Research Article

[9] Wanjiru, L. (2025). Securing IoT Devices: AI and Blockchain as a Dual Defense Mechanism. Algora, 2(2), 53-78.

[10] Mazumder, P. T. (2026). Explainable and fair anti-money laundering models using a reproducible SHAP framework for financial institutions. Discover Artificial Intelligence.

[11] Huang, Y. W., Yu, F., Hang, C., Tsai, C. H., Lee, D. T., & Kuo, S. Y. (2004). Securing web application code by static analysis and runtime protection. *Proceedings of the 13th International Conference on World Wide Web*, 40–52. https://doi.org/10.1145/988672.988679

[12] Sridharan, M., Dolby, J., Chandra, S., Schäfer, M., & Tip, F. (2012). Correlation tracking for points-to analysis of JavaScript. In *European Conference on Object-Oriented Programming (ECOOP)* (pp. 435–458). Springer.

[13] Barua, S. (2025). Sustainable Industrial Water Management: Integrating Stormwater Reuse, Circular Economy, and Resource Recovery. British Journal of Environmental Studies, 5(3), 08-22.

[14] Takon, A. (2025). Explainable AI for Threat Modelling and Decision Support in Engineering Assets. *Journal of Cyber-Physical Security and Robotics*, *1*(02), 46-52.

[15] Li, L., Bartel, A., Klein, J., Le Traon, Y., Arzt, S., & Rasthofer, S. (2014). I know what leaked in your pocket: Uncovering privacy leaks on Android apps with static taint analysis. *arXiv preprint arXiv:1404.7431*.

[16] Allen, N., Gauthier, F., & Jordan, A. (2021). IFDS taint analysis with access paths. *arXiv preprint arXiv:2103.16240*.

[17] Muralee, S., Koishybayev, I., Nahapetyan, A., et al. (2023). ARGUS: A framework for staged static taint analysis of GitHub workflows and actions. *USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity23/presentation/muralee

[18] Takon, A. (2026). AI-Augmented Visual Inspections in Mining and Heavy Industry. *Journal of Science Technology and Social Transformation*, *2*(01), 8-16.

[19] Kola, J. N. (2023). Quantifying Revenue Impact of Enterprise Analytics: A Revenue Attribution Framework for Business Intelligence Systems.

[20] Takon, A. (2023). Machine Learning (ML)–Based Cyber Threat Modelling for Industrial Control Systems in critical Infrastructure. *International Journal of Technology, Management and Humanities*, *9*(02), 94-108.

[21] 5. Njuguna, L. (2026). Cybersecurity for Small Businesses: Cost-Effective AI-Driven Solutions. CogNexus, 2(1), 19-40.

[22] Singh, S. S. (2023). Code Compliance Challenges in High-Stakes Infrastructure Projects. *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology*, *15*(01), 213-221.

[23] Williams, M. O. (2024). DEVELOPMENT OF REACTIVE HEAT EXCHANGERS FOR ENHANCED GEOTHERMAL ENERGY RECOVERY. Power System Protection and Control, 52(2), 18-37.

Research Article

[24] Kola, J. N. (2023). Measuring the Business Value of Analytics-Driven Decisions: A Decision Impact Attribution Framework for Enterprise Environments.

[25] Singh, S. S. (2023). Architectural Identity in Transit Infrastructure: Branding vs Functionality. *Multidisciplinary Innovations & Research Analysis*, *4*(2), 1-12.

[26] Mazumder, P. T. (2025). Blockchain in trade finance: reducing fraud and improving efficiency through digital ledger technology. Digital Finance, 7(4), 1043-1063.

[27] Njuguna, L. W. (2026). Deepfake Cybersecurity Threats: Detection and Mitigation Strategies. International Journal of Artificial Intelligence and Engineering Research, 2(01).

[28] Singh, S. S. (2023). Human-Centered Design in Underground Transit Environments. *Multidisciplinary Innovations & Research Analysis*, *4*(3), 1-20.