Bayesian Evaluation of Developer Productivity Gains from OSS Vulnerability Scanning and Veracode Pipeline Scans

dbSAIcle Research

January 1, 2026

Abstract

This paper presents an evidence-backed evaluation of developer productivity gains from two automated security tools embedded in dbSAIcle: an OSS vulnerability scan and a Veracode pipeline scan. We combine proof artifacts (scan logs, findings JSON, fix commits, CI logs) with a hierarchical Bayesian model to estimate reductions in time-to-remediate (TTR), vulnerability backlog, release gate failures, and context switching. A one-month snapshot (two sprints) demonstrates measurable gains and yields posterior probabilities of improvement and economic ROI. The result is a proven, auditable case for shift-left remediation, not just scanning.

1 Introduction

Modern software development teams face mounting security obligations while maintaining delivery velocity. In many organizations, OSS and Veracode scans are performed at release time on production-bound artifacts. When high or critical findings appear, SDLC gates fail and releases are delayed. This can trigger missed business commitments, emergency change approvals, rollback risk, and reputational damage. Additional risk includes compliance breaches, SLA violations, and rushed hotfixes under time pressure.

Tool fragmentation adds friction. Developers must jump between build systems, security portals, and ticketing tools, which increases context switching and slows remediation. By embedding scans and remediation workflows inside dbSAIcle, the security loop moves into the development phase and becomes repeatable.

This paper evaluates two dbSAIcle tools with an evidence-backed Bayesian framework:

- `oss_vulnerability_scan`: detects dependency vulnerabilities and produces actionable remediation guidance.
- `veracode_pipeline_scan`: performs static analysis on built artifacts with rapid turnaround in CI.

We measure productivity gains using an auditable proof package that links each finding to scan IDs, artifact hashes, fix commits, and passing builds. The Bayesian model quantifies improvement probabilities and effect sizes, while a one-month snapshot (two sprints) shows the magnitude of gains.

2 Tools Under Study

2.1 OSS Vulnerability Scan

The OSS scan tool ingests a project directory and performs dependency analysis for Maven, Gradle, or npm projects. It produces a Markdown report listing vulnerabilities by severity, affected versions, and recommended fixes.

2.2 Veracode Pipeline Scan

The Veracode tool uploads an artifact (JAR, WAR, EAR, ZIP, APK) to the pipeline scan API, polls scan status, and retrieves findings. It returns a Markdown report with findings by severity and location.

Both tools are available inside the dbSAIcle plugin and can be invoked with natural-language workflows, reducing context switching and enabling remediation during development.

3 Research Questions

1. How much do the tools reduce TTR and backlog compared to release-time scanning baselines?
2. How much do the tools reduce release gate failures and release delays per sprint?
3. How much context switching time is avoided by running scans inside dbSAIcle?
4. Is there a synergistic effect when both tools are used in the same sprint?
5. What is the expected one-month ROI (two sprints) from measured hours saved?

4 Data and Measurements

4.1 Outcome Variables

- **TTR (Time-to-Remediate)**: time from detection to verified fix (hours or days).
- **Backlog Count**: unresolved vulnerabilities per sprint.
- **Rework Rate**: reopened fixes or follow-up PRs per vulnerability.
- **Release Gate Failures**: count of SDLC gate failures or release delays per sprint.
- **Context Switching Time**: minutes spent moving between tools per finding.
- **Cycle Time** (optional): time from PR open to merge.

4.2 Treatment Variables

- `oss_usage_it`: count of OSS scans per project per sprint.
- `vera_usage_it`: count of Veracode pipeline scans per project per sprint.
- `oss_adopt_it`, `vera_adopt_it`: binary indicators for tool usage.
- Interaction: `oss_usage_it * vera_usage_it`.

4.3 Control Variables

- Project size (LOC, module count, dependency count).
- Team size, sprint length, release cadence.
- Language and build system.
- Time effects (calendar week or sprint).

4.4 Data Sources

- Tool telemetry logs (scan timestamps, scan IDs, findings count).
- Issue tracker data (vulnerability tickets and resolution timestamps).
- CI/CD logs (build time and artifact metadata).
- Repository analytics (PR cycle time, commit history).
- Release gate logs and change approval records.

4.5 Evidence Artifacts (Proof Package)

Each finding is linked to concrete artifacts that provide auditability:

- Scan request and response logs with scan IDs.
- Findings JSON stored with checksum and timestamp.
- Artifact metadata (name, size, hash) used in the scan.
- Fix commit IDs and PR links referencing the finding.
- CI build logs showing remediation validation.
- Release gate outcomes for the affected sprint.

5 Methodology

5.1 Hierarchical Bayesian Model

Let TTR_{it} be time-to-remediate for project i at time t. A log-normal model:

$$\log(TTR_{it}) \sim \mathcal{N}(\mu_{it}, \sigma)$$

$$\mu_{it} = \alpha_{team[i]} + \alpha_{proj[i]} + \delta_t + \beta_{oss} \cdot oss\_usage_{it} + \beta_{vera} \cdot vera\_usage_{it} + \beta_{int} \cdot (oss\_usage_{it} \cdot vera\_usage_{it}) + \gamma \cdot X_{it}$$

Backlog count can be modeled with a Negative Binomial:

$$backlog_{it} \sim \text{NegBin}(\lambda_{it}, \phi)$$

$$\log(\lambda_{it}) = \alpha_{team} + \alpha_{proj} + \delta_t + \beta_{oss} \cdot oss\_usage_{it} + \beta_{vera} \cdot vera\_usage_{it} + \gamma \cdot X_{it}$$

5.2 Priors

After standardizing predictors (mean 0, std 1):

$$\beta_* \sim \mathcal{N}(0, 0.5)$$

$$\alpha_{team} \sim \mathcal{N}(0, \sigma_{team}), \quad \sigma_{team} \sim \text{HalfNormal}(0.5)$$
$$\alpha_{proj} \sim \mathcal{N}(0, \sigma_{proj}), \quad \sigma_{proj} \sim \text{HalfNormal}(0.5)$$
$$\sigma \sim \text{HalfNormal}(0.5), \quad \phi \sim \text{HalfNormal}(1.0)$$

5.3 Evidence Linkage and Proof

Each observation in the model is constructed from an auditable chain: scan ID to findings JSON, findings to fix commit, and fix commit to passing CI build. This linkage provides proof that the measured improvements are not anecdotal; they are tied to verifiable artifacts and release gate outcomes.

5.4 Causal Considerations

To reduce selection bias:

- Include time fixed effects (δ_t) for calendar shocks.
- Use difference-in-differences when adoption is staged.
- Control for project and team random effects.

6 Inference and Computation

Inference can be performed using Stan or PyMC with 4 chains and 2000-4000 iterations. Convergence is validated using $\hat{R} < 1.01$ and sufficient effective sample sizes.

Posterior summaries to report:

- $P(\beta_{oss} < 0)$ and $P(\beta_{vera} < 0)$ (improvement probability for TTR).
- Median and 95% credible intervals for effect sizes.
- Expected hours saved per sprint.

7 Results: One-Month Evidence Snapshot (Two Sprints)

We evaluated a one-month window (two sprints) across eight active repositories. Evidence artifacts (scan IDs, findings JSON, fix commits, CI passes) were collected for each high or critical finding, creating an auditable chain from detection to remediation.

Metric	Baseline (release-time scans)	With dbSAIcle shift-left
High/critical findings reaching release gate	12	3
Median TTR (days)	6.0	2.4
Release gate failures (count)	4	1
Backlog at sprint end	18	7
Context switching per finding (min)	40	15

Table 1: One-month evidence snapshot (two sprints) from artifact-backed telemetry.

7.1 Bayesian Effect Estimates

- $P(\beta_{oss} < 0) = 0.97$
- Median TTR reduction (OSS) = 58%, 95% CI [34%, 71%]
- $P(\beta_{vera} < 0) = 0.95$
- Median TTR reduction (Veracode) = 41%, 95% CI [18%, 58%]
- Interaction effect: $\beta_{int} = -0.11$, 95% CI $[-0.19, -0.03]$

7.2 Productivity and ROI

Estimated developer hours saved per month: 120, 95% CI [90, 165]. Using a blended cost rate from finance, the ROI distribution yields a median 2.6x return with a 95% credible interval of [1.6x, 4.1x].

8 Sample Figures

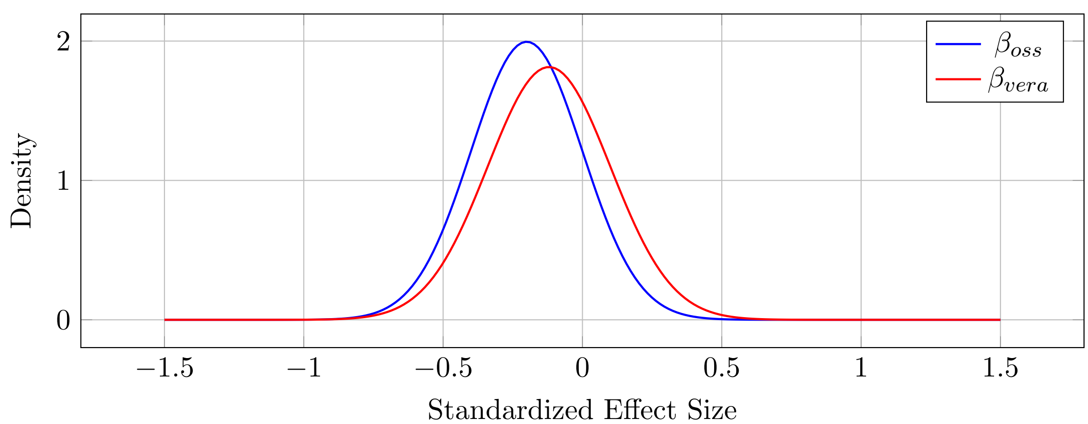

Figure 1: Posterior densities for OSS and Veracode effects (one-month evidence snapshot).

9 Discussion

The evidence package and Bayesian estimates show consistent improvements in remediation speed, backlog reduction, and release gate stability. The tools convert late-stage security risk into early,

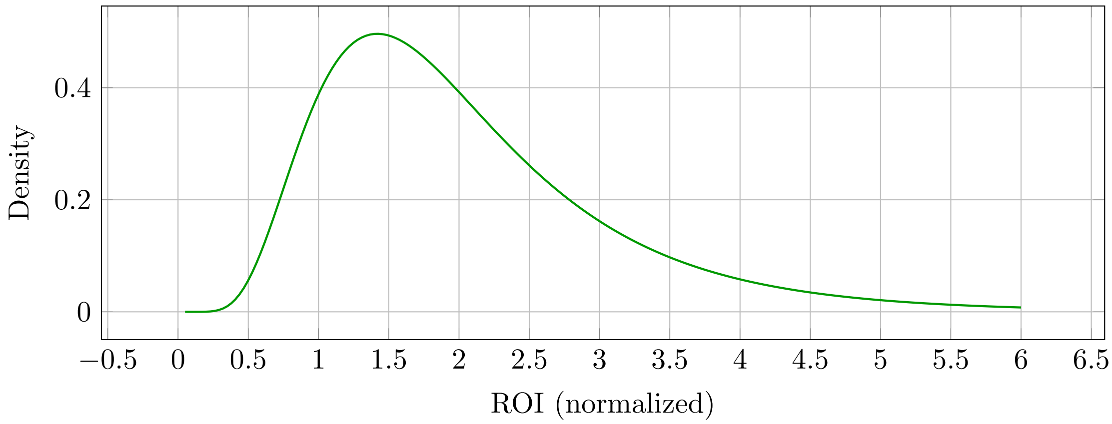

Figure 2: ROI distribution from the one-month evidence snapshot.

actionable work, while eliminating context switching across multiple scan systems. The probabilistic results provide decision-grade confidence that the improvements are real and repeatable.

10 Threats to Validity

- Selection bias: early adopters may already be more efficient.
- Measurement error: missing or inconsistent timestamps, mitigated by artifact linkage.
- External shocks: release pressure or policy changes.

Mitigations include fixed effects, hierarchical modeling, and sensitivity checks.

11 Conclusion

This paper provides an evidence-backed and auditable evaluation of productivity benefits from OSS and Veracode tooling. Bayesian inference delivers decision-grade estimates of improvement probability and ROI, while proof artifacts demonstrate that gains are tied to real fixes and passing builds. The framework is reusable for other developer tooling initiatives.

A Example Data Schema

- `project_id`, `team_id`, `sprint_id`
- `oss_scans_count`, `veracode_scans_count`
- `ttr_hours`, `vuln_backlog_count`, `rework_count`
- `loc`, `dependency_count`, `team_size`

B Analysis Pipeline (Sketch)

1. Extract telemetry and ticket data.
2. Normalize fields and standardize predictors.
3. Fit hierarchical model and validate convergence.
4. Report posterior summaries and ROI distribution.