International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group | Volume 5, Issue 4, 175-179, 2024
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V5I4P118

Original Article

Shift-Left Security for Decentralized Engineering Organizations: Embedding SAST, DAST, and Penetration Testing Throughout the Software Development Lifecycle in University and Research Computing Environments

Sri Gantikota
Senior Software Engineer, San Diego, California 92101, USA.

Abstract - *Shift-left security in a single product engineering organization is a well-documented practice. Adapting the same patterns to a decentralized engineering organization, in which multiple independently governed teams ship software on heterogeneous timelines using heterogeneous tooling, requires additional attention to consent, autonomy, and the operational realities of cross-team coordination. University research computing environments are a representative instance of this organizational shape. This paper describes the design and rollout of a shift-left security program in a decentralized higher-education engineering setting. The program embeds Static Application Security Testing, Dynamic Application Security Testing, and penetration testing throughout the software development lifecycle, but it does so through invitation and incentive rather than through top-down mandate. The paper covers the rationale for the design choice, the architecture of the central platform components that teams opt into, the developer-facing artifacts that lower the cost of adoption, the integration patterns with the heterogeneous continuous integration systems in use across teams, and the metrics that track program reach and effectiveness over time. The paper closes with a discussion of how the same patterns transfer to other decentralized engineering organizations such as large enterprises with federated product groups, scientific computing collaborations, and open-source project ecosystems. The intent is to document a practical approach that engineering organizations whose authority structure does not support a security mandate can use to nonetheless drive substantive improvements in their security posture.*

Keywords - *Shift-Left Security, Devsecops, Decentralized Organization, Higher Education, Research Computing, SAST, DAST, Penetration Testing, SDLC, Opt-In Security Platform, Federated Engineering*.

1. Introduction

Shift-left security is the practice of moving security activities earlier in the software development lifecycle so that vulnerabilities are detected and remediated closer to the point at which they are introduced. The practice is widely documented in the context of single product engineering organizations, in which a single leadership team can mandate the adoption of tooling and the operation of processes. The practice is less well-documented in the context of decentralized engineering organizations, in which authority is distributed across multiple teams that operate on independent timelines and that retain the right to choose their own tooling.

University research computing is a representative instance of the decentralized shape. A typical research university has dozens or hundreds of engineering teams across academic departments, administrative units, and research groups. Each team has its own leadership, its own funding source, and its own delivery cadence. A central information security office exists, but its authority over how a given team writes code is limited. A security program that requires every team to adopt a specific tool on a specific timeline will not succeed in this setting, because the central office does not have the authority to enforce such a requirement and the teams have legitimate reasons to make their own choices.

This paper describes the design and rollout of a shift-left security program adapted to this organizational shape. The program embeds Static Application Security Testing, Dynamic Application Security Testing, and penetration testing throughout the software development lifecycle through invitation and incentive rather than through mandate. The contribution of the paper is the design rationale, the architecture of the opt-in central platform, the developer-facing artifacts that lower the cost of adoption, and the metrics that track program reach and effectiveness.

The rest of the paper is organized as follows. Section 2 describes the organizational context and why mandate-based approaches do not transfer. Section 3 describes the program architecture and its opt-in design. Section 4 covers the developer-facing artifacts that drove adoption. Section 5 covers integration with heterogeneous continuous integration systems. Section 6 describes the metrics framework. Section 7 reports observations from the rollout. Section 8 discusses transferability to other decentralized organizations. Section 9 concludes.

2. Organizational Context

2.1. Decentralized Authority

A research university distributes authority across academic schools, departments, research centers, and administrative units. Each unit has its own budget, its own staffing, and its own delivery priorities. A central information security office establishes policies, but the operational decisions about how a specific application is built and deployed are made by the unit that owns the application. This decentralization is not an organizational defect; it reflects the academic principle that scholarly units operate with substantial autonomy.

2.2. Heterogeneous Tooling

Tooling is heterogeneous as a consequence of the decentralization. Different teams use different continuous integration systems, different source control hosts, different deployment platforms, and different programming languages. The heterogeneity is not a problem the central office can solve by standardizing tooling, because standardization would require the same authority that the office does not have. The security program therefore has to operate across heterogeneity rather than reduce it.

2.3. Why Mandate Does Not Transfer

A shift-left security program that depends on a mandate cannot operate in this setting. The central office can issue a policy that says SAST is required, but it cannot enforce the policy across teams it does not control. Teams that find the mandated tool inconvenient will work around it, or comply nominally without the underlying security benefit. The result of a mandate-based program in this setting is a paper compliance posture without the substantive security improvement the program was supposed to deliver.

2.4. Why Invitation Works

Invitation-based programs work in this setting because they align the program\'s success with the team\'s success. A team adopts a tool because it reduces the team\'s own work, not because the team has been ordered to. The program\'s task is to make adoption easy enough and the benefit obvious enough that adoption is rational from the team\'s point of view. This is a different design problem than mandate-based security, but it is a soluble one.

3. Program Architecture

3.1. Opt-In Central Platform

The central platform provides shared infrastructure that teams can opt into. The infrastructure includes a managed SonarQube instance that teams can point their continuous integration runs at, a DAST scanning service that teams can submit their staging URLs to, a penetration testing scheduling service that teams can book into, and a findings management portal that aggregates results across the tools. Each component is operated by the central office and is free at the point of use for any team that opts in.

3.2. Self-Service Onboarding

Onboarding a project to the platform is self-service. A team creates an account, registers a project, points their continuous integration runs at the SonarQube instance, and starts seeing findings. There is no central queue, no approval workflow, no mandatory training before the first scan. The friction of starting is deliberately kept low because friction is the dominant variable in adoption.

3.3. Tiered Engagement Levels

The platform supports tiered engagement levels. The lightest level is SAST-only with self-service triage. The intermediate level adds DAST and a quarterly review by the central security team. The heaviest level adds scheduled penetration testing and remediation co-engineering with the central team. Teams choose their level based on the criticality of their application and the engineering capacity they can allocate. The tiered structure lets a team start light and increase engagement as the value becomes evident.

3.4. No Mandatory Findings Sharing

Findings produced for a team belong to the team. The platform does not share findings outside the team without the team\'s consent. This is critical for adoption because teams will not opt into a platform that exposes their security posture to other teams or to the central office without their agreement. The aggregate metrics the central office uses to track program effectiveness are computed in a way that does not require per-project disclosure of findings.

4. Developer-Facing Artifacts

4.1. Quickstart Guides

Quickstart guides cover the common integration paths. There is a guide for SonarQube with Jenkins, with Bamboo, with GitHub Actions, and with GitLab pipelines. There is a guide for DAST against a staging URL. There is a guide for booking

penetration testing. Each guide is short and concrete, with the smallest set of steps needed to get a first scan running. Comprehensive documentation is available separately for teams that want it, but the quickstart guides are deliberately scoped to the first day of adoption.

4.2. Reference Pipelines

Reference pipelines are provided as version-controlled configurations that teams can fork. The reference pipelines cover the typical project shapes the platform supports: a Java back-end with a JavaScript front-end, a Python service, a containerized microservice, and a static site. A team that matches one of these shapes can fork the reference, adjust the project-specific values, and have a working pipeline in less than an hour. The reference pipelines are maintained by the central office and kept current with the tooling versions in production.

4.3. Office Hours and Pair Sessions

The central security team holds open office hours each week. Any team can drop in with questions about the platform or about specific findings. The format is informal and the questions range from how to interpret a finding to how to convince leadership to fund remediation work. Pair sessions are also available for teams that want hands-on help on a specific issue. The office hours and pair sessions are the channel through which a substantial fraction of platform onboarding actually happens, because the personal interaction is what gets a hesitant team across the activation threshold.

4.4. Curated Finding Catalog

A curated catalog of common findings, with worked examples and remediation patterns, is published as a reference. The catalog covers the OWASP Top Ten categories and the additional categories that recur across the projects the platform serves. Each entry describes the underlying vulnerability, gives an example of vulnerable code, gives an example of remediated code, and explains how the platform\'s tools detect it. The catalog is the resource a developer reaches for when they have a finding they do not immediately understand.

5. Integration with Heterogeneous Continuous Integration

5.1. Common Adapter Layer

The platform exposes a common adapter layer that abstracts over the differences between the continuous integration systems in use across teams. A team integrating their pipeline calls the same interface regardless of whether they are running Jenkins, Bamboo, GitHub Actions, GitLab pipelines, or a homegrown system. The adapter layer handles the system-specific details of triggering scans, reporting status, and posting comments.

5.2. Pull-Request Feedback Where Available

Where the source control host supports pull-request comments, the platform posts findings as inline comments. This is the form of feedback developers find most actionable, because it appears in the workflow they are already using. Hosts that do not support pull-request comments fall back to email notification, with a deep link into the findings portal. The pull-request path is the preferred integration but is not required.

5.3. Asynchronous DAST Submission

DAST scans are submitted asynchronously. A team submits a staging URL and a scan profile, and the platform returns a job identifier. The team is notified when the scan completes. The asynchronous design is necessary because DAST scans take longer than the typical continuous integration build is willing to wait for, and synchronous integration would either slow builds unacceptably or force a partial scan that misses findings.

6. Metrics Framework

6.1. Reach Metrics

Reach metrics track the number of projects opted into the platform, the number of scans performed, and the number of teams engaged at each tier. Reach is the leading indicator of program effectiveness because findings cannot be remediated on projects that have not opted in. The reach metrics are reported to the central office leadership and to the broader engineering community as a transparency signal.

6.2. Findings and Remediation Metrics

Findings and remediation metrics are computed in aggregate without per-project disclosure. The aggregate metrics include the total volume of findings by severity, the median time to remediate by severity, and the trend over time. The aggregate form is what the central office uses to demonstrate program effectiveness while preserving the per-project confidentiality that adoption depends on.

6.3. Per-Team Reports

Each team receives a per-team report that shows their own findings, their remediation rate, and their position in the aggregate distribution. The per-team report is private to the team and is structured to support the team\'s internal review rather than external comparison. Teams that want to share their report more widely can do so, but the default is private.

6.4. Penetration Testing Outcomes

Penetration testing outcomes are reported separately because the cadence is different. Each engagement produces a report that the engaged team receives in full and that the central office sees in aggregated form. The aggregate trend across engagements is reported to leadership as an indicator of the maturity of the program.

7. Rollout Observations

7.1. Early Adoption

Early adoption was driven by teams that already had security on their roadmap and were looking for a path of least resistance. These teams were the easiest to bring into the platform because they had decided to adopt some form of shift-left security and were open to the platform as the vehicle. Their adoption produced the first wave of aggregate metrics that the central office could use to demonstrate the program\'s value to less-engaged teams.

7.2. Adoption through Referral

A substantial fraction of subsequent adoption came through referral. A team that had adopted the platform would mention it to a peer team facing a similar problem, and the peer team would self-onboard. The referral pattern is what allowed the program to grow without proportionate growth in the central office\'s outreach effort. The pattern depends on the platform actually working well for the teams that adopt it; a platform that underdelivers will not produce positive referrals.

7.3. Persistent Holdouts

Some teams did not adopt the platform during the period in scope. The reasons varied. Some teams had limited engineering capacity and could not absorb even the small adoption cost. Some teams had specialized tooling needs that the platform did not cover. Some teams had organizational reasons for keeping security work internal. The program does not treat persistent holdouts as failures; the design choice to make adoption invitation-based means accepting that some teams will not invite themselves at any given time.

7.4. The Role of Compliance

Compliance requirements provided a useful adjacent driver. Some teams adopted the platform because an external compliance regime they were subject to required documented security testing. The platform\'s reports were structured to satisfy these compliance regimes, which made adoption a single move that addressed both the security goal and the compliance obligation. The convergence of the two is a recurring pattern in the data.

8. Transferability

8.1. Large Enterprises with Federated Product Groups

Large enterprises with federated product groups have a similar organizational shape to the research university described here. Different product groups have different leadership, different tooling, and different release cadences. A central security office can adopt the same invitation-based model with appropriate adjustments for the enterprise\'s culture. The adjustments include the use of executive sponsorship to amplify the program\'s visibility and the integration of program metrics into enterprise-wide engineering scorecards.

8.2. Scientific Computing Collaborations

Scientific computing collaborations, such as those that operate large physics or astronomy experiments, have a strongly decentralized character because the participating institutions are independent. The invitation-based model fits naturally because the collaboration cannot mandate practices across institutions. The platform\'s role is to provide shared infrastructure that any participating institution can opt into, with results visible only to the institution that produced them.

8.3. Open-Source Project Ecosystems

Open-source project ecosystems are an even more decentralized form. A central foundation cannot mandate the practices of independent projects in its ecosystem, but it can provide shared infrastructure and documentation that lower the cost of adoption. The patterns described in this paper apply, with adjustments for the volunteer character of the engineering work in many open-source projects.

9. Conclusion

Shift-left security in a decentralized engineering organization requires a different design approach than in a single product engineering organization. The mandate-based approach that works in the single-product setting does not transfer because the central office does not have the authority to enforce a mandate across teams it does not control. An invitation-based approach,

in which a central platform provides shared infrastructure that teams opt into, can produce substantive security improvements without the authority that mandate-based approaches require. The design depends on lowering the friction of adoption, on respecting the autonomy of the adopting teams, and on producing aggregate metrics that demonstrate program effectiveness without violating per-project confidentiality. The patterns transfer to other decentralized engineering organizations including large enterprises with federated product groups, scientific computing collaborations, and open-source project ecosystems. Security programs operating in similar organizational contexts can use this paper as a starting point for their own designs.

Acknowledgments

This work was performed in the context of security and cross-functional collaboration at the University of California, San Diego. The author thanks the central DevOps and Product Security teams and the engineering teams across the university who have engaged with the program.

Conflicts of Interest

The author declares that there is no conflict of interest concerning the publishing of this paper.

References

[1] Open Web Application Security Project. OWASP Top Ten Web Application Security Risks, 2021 edition. https://scholar.google.com/scholar?hl=en&q=OWASP Top Ten Web Application Security Risks, 2021 edition

[2] SonarSource. SonarQube static analysis platform documentation. https://scholar.google.com/scholar?hl=en&q=SonarQube static analysis platform documentation

[3] International Business Machines Corporation. IBM Security AppScan dynamic application security testing documentation. https://scholar.google.com/scholar?hl=en&q=IBM Security AppScan dynamic application security testing documentation

[4] National Institute of Standards and Technology. Secure Software Development Framework, NIST Special Publication 800-218. https://scholar.google.com/scholar?hl=en&q=Secure Software Development Framework, NIST Special Publication 800-218

[5] National Institute of Standards and Technology. Application Security Verification Standard, OWASP ASVS 4.0.3. https://scholar.google.com/scholar?hl=en&q=Application Security Verification Standard, OWASP ASVS 4.0.3

[6] Open Web Application Security Project. OWASP Software Assurance Maturity Model, version 2.0. https://scholar.google.com/scholar?hl=en&q=OWASP Software Assurance Maturity Model, version 2.0

[7] Building Security In Maturity Model. BSIMM annual industry report, Synopsys. https://scholar.google.com/scholar?hl=en&q=BSIMM annual industry report, Synopsys

[8] National Institute of Standards and Technology. Framework for Improving Critical Infrastructure Cybersecurity, NIST Cybersecurity Framework 2.0. https://scholar.google.com/scholar?hl=en&q=Framework for Improving Critical Infrastructure Cybersecurity, NIST Cybersecurity Framework 2.0

[9] National Institute of Standards and Technology. Security and Privacy Controls for Information Systems and Organizations, NIST Special Publication 800-53 Revision 5. https://scholar.google.com/scholar?hl=en&q=Security and Privacy Controls for Information Systems and Organizations, NIST Special Publication 800-53 Revision 5

[10] EDUCAUSE. Higher Education Information Security Council resources and annual security survey. https://scholar.google.com/scholar?hl=en&q=Higher Education Information Security Council resources and annual security survey

[11] Internet2. Trust and Identity in Education and Research community documentation. https://scholar.google.com/scholar?hl=en&q=Trust and Identity in Education and Research community documentation

[12] United States Office of Management and Budget. Memorandum M-22-09, Moving the US Government Toward Zero Trust Cybersecurity Principles. https://scholar.google.com/scholar?hl=en&q=Memorandum M-22-09, Moving the US Government Toward Zero Trust Cybersecurity Principles

[13] Cybersecurity and Infrastructure Security Agency. Secure by Design and Default principles. https://scholar.google.com/scholar?hl=en&q=Secure by Design and Default principles

[14] Atlassian. Bamboo continuous integration server documentation. https://scholar.google.com/scholar?hl=en&q=Bamboo continuous integration server documentation

[15] Jenkins project. Jenkins user documentation. https://scholar.google.com/scholar?hl=en&q=Jenkins user documentation

[16] GitHub. GitHub Actions documentation. https://scholar.google.com/scholar?hl=en&q=GitHub Actions documentation

[17] GitLab. GitLab CI/CD documentation. https://scholar.google.com/scholar?hl=en&q=GitLab CI/CD documentation

[18] United States Department of Education. Family Educational Rights and Privacy Act, 20 U.S.C. 1232g. https://scholar.google.com/scholar?hl=en&q=Family Educational Rights and Privacy Act, 20 U.S.C

[19] United States Department of Health and Human Services. Health Insurance Portability and Accountability Act Security Rule, 45 CFR Part 164 Subpart C. https://scholar.google.com/scholar?hl=en&q=Health Insurance Portability and Accountability Act Security Rule, 45 CFR Part 164 Subpart C