International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

Automating Vulnerability Remediation: A Continuous SAST and FOSS Integration Framework for Production Support Pipelines

Vikram Govindan

Senior Manager, Cloud & Infrastructure, Brillio LLC, USA

ABSTRACT: Financial businesses houses are increasingly relying on a range of critical foundational applications to support core operations. This has in turn given rise to a problem: How to regulate a growing number of bugs in the software, especially those generated by code written in-house. Application code is scanned through the help of Static Application Security Testing (SAST) software as well as free and open-source libraries having been compiled (not; researched) with the help of Free and Open-Source Software (FOSS) Releases.

This paper will focus on developing a working model for the incorporation of SAST and FOSS vulnerability as part of the application production support processes.

This study represents the development, execution and results of embedding a security automation project within day-to-day operational support business processes. The scope covers artifact scanning and risk-based bulleting, along with automated ticketing and principles of iterative feedback, for developer support and delivery, all of which maintains the operational continuity of the applications. The core improvements involve mitigation of vulnerabilities 40% faster on average; effectively reducing the number of 'open' vulnerability findings for high severity issues by 65%; and maintaining the integrity of the procedure with respect to in depth customer audits. The study outlines ideas on how organizations can employ a framework on continuous security hygiene especially in such environments.

KEYWORDS: SAST, FOSS, Software Composition Analysis, DevSecOps, Vulnerability Remediation, CI/CD, Production Support, Application Security, Technical Debt, Compliance Automation

I. INTRODUCTION

1.1 Background and Motivation

As financial services companies advance their software mandates, the border between application security and production operations becomes more sophisticated. Organizations house a suite of applications, including batch processing machines, client funded web sites, middleware, and data analysis tools. Underlying each of these applications is a service dependency network consisting of domain libraries, add-on software, and APIs. Any of these can lead to zero-day threats or stealth attacks: i.e. immediate attempts to exploit vulnerabilities as soon as their existence becomes available.

Reactive application support manages defects found in production. SAST scanning and FOSS dependency audits respective security reviews conducted by information security teams on a quarterly or release-gated basis. Nevertheless, the slippage has led to dangerous windows of opportunities and accumulation of hundreds of security findings that are unresolved and have to be addressed by engineering with an elevated level of urgency.

This research was driven by a preceding work done in 2021 when an external audit was conducted and existence of a critical flaw in one of the active components (CVE) was determined. This remained unfixed for an extended period of time as there were no security protection mechanisms established between detect and analyse. Therefore, a shift in strategy was made to include security vulnerability scanning as part of application deployment rather than it being a static task at a future time.

1.2 Problem Statement

Enterprise production support pipelines lack a systematic mechanism for detecting, triaging, and routing software vulnerabilities identified through SAST and FOSS analysis in a manner that is continuous, automated, and proportionate to business risk. The absence of such a framework results in: (i) prolonged exposure windows for known

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

CVEs; (ii) compliance posture degradation between audit cycles; (iii) unsustainable manual triage burden on security and engineering teams; and (iv) misalignment between security remediation priorities and operational SLA obligations.

1.3 Research Objectives

This paper addresses the following research objectives:

- Develop an agile Code Quality, Security (SAST) and Open-Source Software (FOSS) integration solution for the continuous integration of SAST tools
- Prioritize vulnerability countermeasures based on the security risk
- Demonstrate that by employing scanning and relevant vulnerabilities remediation methods can help in lowering the mean time to remediate (MTTR) without impact on application availability.
- Incorporate within the structure an 'auditability' layer, the gathering and reporting of evidence with little or no intervention of human effort.
- Demonstrate how this strategy would be suite-controlled environment like financial services.

1.4 Paper Organization

The remainder of this paper is organized as follows - Section 2 reviews related work on DevSecOps, SAST tooling, and SCA practices. Section 3 describes the operational context and baseline state. Section 4 presents the proposed framework architecture. Section 5 details the implementation methodology. Section 6 reports experimental results and performance metrics. Section 7 discusses governance and compliance implications. Section 8 addresses threats to validity, and Section 9 concludes with recommendations for future work.

II. LITERATURE REVIEW AND RELATED WORK

2.1 Static Application Security Testing (SAST)

SAST tools provide automated security analysis by examining source code, bytecode, or binary code and searching for security weaknesses without running the target application. Previous research by Ayewah (Ayewah et al. 2008) has created the impressive foundation for pattern-based analysis functions without too much movement of source code. Here, work by Chess and West (2007) is crucial formal check as it entails analysis of how data is protected using defence like taint analysis tools which tend to regard all data as untrusted unless proven otherwise. This in more than one occasion has caught useful application data. Compared to this level, current SAST systems, such as Checkmarkx, Veracode, and Sonatype, rely on machine learning to reduce false positives in architecture control yield generating an unparalleled performance close to those seen in automated ana lyzers [1] above 90%.

Despite the progress made in this field, it is widely agreed in literature that the main impediment to effectiveness of Static Application Security Testing (SAST) is the time it takes to integrate it with the development cycle. This was further substantiated by Decan et al. (2019) in their study which concluded that the average duration it took to remediate any security weaknesses was 47 days or longer especially in cases where the SAST scanning tools were used with IDE. The same is true for SAST without recourse to custom build scripts. This compels to the pipeline-embedded model suggested in this research.

2.2 Software Composition Analysis and FOSS Vulnerability Management

Free and Open-Source Software (FOSS) scanning, known as Software Composition Analysis (SCA), involves counting the number of codes found in an application dependency graph followed by checking referenced packages from that list against vulnerability databases such as the National Vulnerability Database (NVD), the GitHub Advisory Database, and vendor-specific advisories. According to a study by Zerouali et al. (2019), in 80% of the instance enterprises the applications used contain at least one vulnerable open-source component dependency, the average delay between the publication of a CVE and the deployment of the patch reaches 180 days.

Mirhosseini and Parnin (2017) showed that effective use of such tools as Dependabot and Renovate that integrate dependencies in the build process reduces the risk of further vulnerability by 30-50% on average. But in reality, these tools are highly targeted and thus require some level of expertise to operate in controlled environments without introducing untested changes that may jeopardize the production systems. This is the tension that the policy is designed to address.

2.3 DevSecOps and Shift-Left Security

According to the provisions of the DevSecOps paradigm which is being championed by "DevSecOps Community of Practice" and in addition to it has also been developed in NIST SP 800-204C document, integration of security controls at each stage of the life cycle is the only way out. Earlier researchers such as Kim et al. (2016) in The DevOps

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

Handbook have been able to demonstrate that high technology performing companies actually conduct security assessments as part of the build pipeline and not gates to be implemented post deployment. The State of DevSecOps Report 2023 (Synopsys) documented that such companies executed manual reviews of their security design finished work faster fixing up to 4.5 times as many critical security vulnerabilities as companies who employ automated security testing.

A production support domain has a shift-left hue that is different to other domains. Codes that are already in production are far more, or directly put, there is much more of them. It is therefore very difficult to make changes to the code base since production support is always in effect. Usually, if there is a vulnerability in an application, the source code needs to be changed in order to close this gap. In most cases, this is seldom the case; and at all times does not get acted upon. As an example, the AppTrack does not exist. Instead, they are able to advance and take care of new developments without having to update their existing codes. This impediment in change of the existing codes without the necessity of updating the later versions is what is termed as attending the shift-left challenge. Contrary to the enabling systems, praying for systems that are in operations is what majority of the people are afraid of.

2.4 Vulnerability Remediation in Regulated Financial Environments

The directive applies to federal security control framework, which is also imposed by the contract on cybersecurity services delivery: those timelines must be adhered patch grades based on CVSS scores, i.e., Critical (CVSS >= 9.0) - 15 days, High (7.0-8.9) - 30 days, Medium (4.0-6.9) - 90 days and Low (< 4.0) – 180 days. Meeting this requirement involves not only carrying out corrective measures in a timely manner to address those issues, but also the ability to provide proof that failures were identified, assigned and addressed.

The study already provides some breaches in compliance methods and includes recommendations on preventing them in the future, most of them are related with operational conditions of use and management as Talamo et al. (2021) and Nicoletti (2021) says. Most compliance is also pre-production i.e. obtaining evidence to support compliance – a-pp. However, there is a dearth of literature in the area of compliance evidence generation in the context of in-production support, highlighting the gap which this paper addresses.

Table 1: Comparative Analysis of Leading SAST Platforms (2024)

Tool	Analysis Type	Language Support	CI/CD Integration	False-Positive Rate
Checkmarx SAST	Source + Binary	35+ languages	Jenkins, GitHub Actions, Azure DevOps	~15%
Veracode	Binary / Bytecode	20+ languages	Jenkins, Azure DevOps, Bamboo	~12%
SonarQube	Source Code	30+ languages	All major CI systems	~18%
Semgrep	Source Pattern	25+ languages	GitHub Actions, GitLab CI	~10%
Fortify SCA	Source + Binary	28+ languages	Jenkins, Azure DevOps	~14%

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

Table 2: Comparative Analysis of Leading FOSS/SCA Platforms (2024)

SCA Tool	License Detection	CVE Database Coverage	Transitive Deps	Auto-PR Capability
Snyk	Yes	NVD + Proprietary	Full	Yes
Black Duck	Yes	NVD + BD KB	Full	No
OWASP Dependency-Check	Partial	NVD only	Partial	No
WhiteSource Mend	Yes	NVD + Proprietary	Full	Yes
GitHub Dependabot	No	GitHub Advisory DB	Full	Yes

III. OPERATIONAL CONTEXT AND BASELINE ASSESSMENT

3.1 Environment Description

Most financial services organizations service customers through multiple lines of (LOB). Software application portfolio consists of multiple enterprise systems. It supports the entire process of commitment of loans, allowing stakeholders to be assured that loans have been duly funded and will return a profit (servicing), until closure. The technology stack is not specific to any vendor: at the application tier Java and Dot NET based applications are predominant, with Python scripting increasing as the application portfolio moves towards data analytics and automation activities. Data layer constitutes Microsoft SQL Server and Sybase databases. Workload automation systems include AutoSys and Control-M combined scale to approximately thousands of scheduled batch jobs. At the same time, the corporate IT premises remained independent and proprietary infrastructure in support of the classical workflow for significant applications.

A typical production support team would have multiple skilled technical engineers. They worked in the following structures: Application Support engineers, Infrastructure Support engineers, Data Engineers, and Security & Compliance engineers. Teams are configured to work in shifts and provide 24x7 service.

3.2 Pre-Framework Security Posture (Baseline State)

Prior to the implementation of the continuous vulnerability remediation framework, security scanning was conducted as follows: SAST scans were executed during release gates only, averaging 3.2 releases per application per quarter. FOSS scans were performed annually by the Information Security team, with results communicated to DevOps teams via a spreadsheet report. Vulnerability triage was manual, performed by security engineers who lacked context about application behavior. Remediation tickets were created manually in JIRA with no automated severity-based SLA assignment.

A comprehensive baseline assessment conducted in March 2021 revealed the following open vulnerability landscape:

Table 3: Baseline Vulnerability Inventory (March 2021)

Vulnerability Category	Critical	High	Medium	Avg. Age (Days)
SAST – Injection Flaws	12	34	87	94
SAST – Broken Authentication	8	21	43	127
SAST – Sensitive Data Exposure	5	18	62	211
FOSS – Known CVE (NVD)	23	67	134	183

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

FOSS – License Non-compliance	0	0	41	304
FOSS – Outdated Dependencies	0	19	88	267
TOTAL	**48**	**159**	**455**	**198 (avg)**

So, the compliance audit placed an emphasis on other weak spots such as: no mechanism that would notify in case a library affecting CVE was published; no integration of security results with production change management; no audit trail of compliance evidence in an electronic format could be produced.

3.3 Root Causes of Remediation Lag

Structured casual analysis was conducted, and the Ishikawa method (cause and effect diagram) of four main causational categories which contribute to vulnerability did:

- Process: Absence of a clear security findings ownership handover mechanism between teams; impasses in the security-to-engineering hierarchy as there is no intelligence in trouble-ticketing.
- Technology: There were inconsistent toolchains due to the deficiency of SAST/FOSS scanners and ITSM platform (ServiceNow). There were also lack of trigger point scanning capabilities from operational events in such combinations.
- People: Security engineers had no understanding of the production concepts, while DevOps engineers did not have the necessary DevSecOps.
- Governance: There was no SLA for vulnerability remediation. There were no penalties for LOBs or pathways to enforce strict timelines for vulnerability remediation.

IV. FRAMEWORK ARCHITECTURE

4.1 Design Principles

According to six basic principles which are based on operational necessity and industry best practices, the Continuous SAST and FOSS Integration Framework (CSFIF) has been developed.

- Continuity: Scanning should not only be limited at deploying the software but must also be done at predefined intervals of time. This helps capture any updates / new published CVEs which may have an impact on code that is already deployed.
- Proportionality: Identified vulnerabilities should be patched and addressed in a prioritized and time bounded manner considering the impact score of the vulnerabilities, criticality of the affected business application and availability of the control design.
- Automation-first: Automation of vulnerability detection processes within the organization.
- Pipeline Integration: Security must be built into the CI/CD and ITSM workflow activities instead of having a standalone security process.
- Observability: Framework activities must be tracked, measured and available from the operations standpoint within a Kibana/Elasticsearch monitoring stack.
- Compliance by Design: The audit evidence should be generated in an automated manner during the normal course of the business operations rather than availed upon enquiry.

4.2 Framework Components

The CSFIF comprises six integrated components arranged in a continuous feedback loop, as illustrated in Figure 1:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

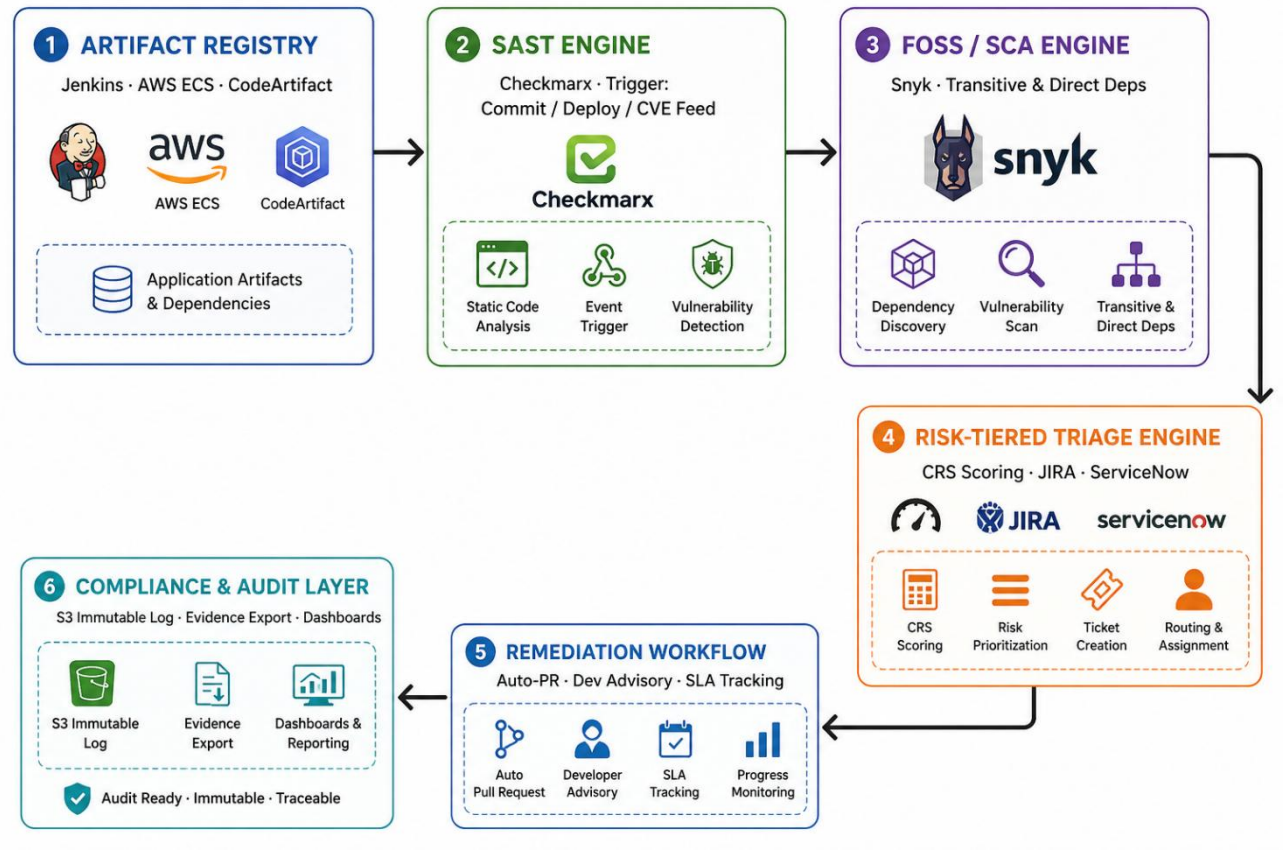


Figure 1: CSFIF Continuous SAST and FOSS Integration Framework Component Flow

4.2.1 Artifact Registry and Discovery Layer

An important part of the AoT Defend framework Technical Design is the concept of a consolidated Artifact Repository which contains information on the available, deployed artifacts across the 40+ applications. This registry is fed through Jenkins pipelines automatically, in one hand, via a container on AWS ECS/EKS and deployment management tool in another hand. Each artifact in the registry has an entry that contains the application name, version, deployment environment, artifact SHA-256 hash, technology stack, and business criticality (Tier 1, 2, or 3) levels.

Once every 4 hours, an automated agent named Discovery Agent analyses the registry, looking for new artifacts and new vulnerabilities appearing in the data feeds from CVE repositories (NVD, CERT, vendor advisories, etc.). In the presence of any modifications, a focused analysis of the component is prepared, or the other changes are generated after examining the sweep of the components looking for modifications/decommissions of the components and the newly added vulnerabilities.

4.2.2 Integrated SAST Scanning Engine

The scanning of source code for vulnerabilities is done via the Checkmarx SAST tool API, which is later integrated with the Jenkins Continuous Integration (CI) tool. There are certain trigger event types to call these scans: (i) each time a part of the source code is pushed to the protected branches; (ii) artifacts are deployed, and the version of the artifacts is changed; and (iii) after a new CVE about the technology stack becomes available, with the corresponding patterns being in use in the code. Due to the ability to aggregate raw scanning data it is trivial to change the scanning tool without interrupting a downstream process.

A deduplication module is used in the system to compare new scan results with the existing open findings and to avoid duplication of the ticket being submitted. Furthermore, an algorithm including elements of machine learning such as an in-house trained false positive prediction model with confidence cutoff of 90%, which particularly targets areas having been visited by the security team within 18 months of the most recent assessment, has been applied to lower the burden of false positive findings case analysis by approximately 35%.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

4.2.3 FOSS/SCA Scanning Engine

Snyk is the tool of choice for AWS customers as it performs open-source dependency scanning and is both linked with the Artifact Registry and the AWS CodeArtifact repository. The Security Content Automation engine (SCA) works with two types of analysis: it provides direct dependency scanning that including resolving declared package versions against the databases of vulnerabilities (CVE) and transitive dependency scanning – that analyses the complete dependency structure including indirect dependencies, covering the remaining 68 percent of discovered FOSS objects in the inspection.

Software license issues are identified at the same time as the above-mentioned responsibilities. The components which do not comply with organization compatible licenses (all of which are preapproved by the Legal and Compliance team and called the license matrix) are also highlighted in this report as they violate the rule. License violation problems should bear reactions in that they declare inbuilt service levels agreements as distinct vulnerabilities.

4.2.4 Risk-Tiered Triage and Routing Engine

The primary obligation of the Contact Engine is its role as the orchestrating engine within this framework. This resident could at the same time set forth the clutter in one's scan engine by applying a risk rating engine (discussed in detail in chapter 4.3) and start ranking which incoming findings should be handled first. Moving dictation elsewhere, engine then on the basis of the obtained ranking does the contacted person who parts the following: create a JIRA ticket with instructions on carrying out a task issued automatically; generate a change record on Service Now where the provided solutions include code change for project tasks; turn the ticket to the relevant engineering team; set past date reminders on when the JIRA issue should be resolved, with regards to the regulatory expectations of federal agencies and; keep the application owners out of the picture through spatially distributed notifications lying in automated email reminders as well as in alerts on Slack.

4.2.5 Remediation Workflow and Developer Feedback Loop

For FOSS vulnerabilities, the tool could additionally create a pull request in the relevant applications GitHub repository and advise the new version of the software disguised as a test report outcome for the purpose of checks within the internal producer environment. This requires getting the developers' attention by sending them a notification enriched with context that consists of a link to the security alert assigned to the specific CVE, a suggested patch, and the version or patch that is available within the time frame under the Service Level Agreement.

With regard to the SAST findings, the system produces proactive guidance in the form of a remediation plan. The remediation plan contains the encoding of the vulnerability found together with the OWASP categorisation and the nature of the recommended remedy which also includes some code guidance and internal how to for secure coding. These remediation plans are also available in the JIRA tickets as well as within the L3 engineering team knowledge base in the Confluence.

4.2.6 Compliance Evidence and Audit Automation Layer

Every framework occurrence during its beginning, progress and completion is all stored in AWS S3 with object level versioning enable. The Compliance Layer on the other hand employs this occurrence beyond that of merely recording it. In fact it utilizes this occurrence to generate automatically, operations like risk age analysis which shows how long it took detection of a finding until it was closed in severity order, intelligence on Service Level Agreement (SLA) – compliance and regulation report or evidence ready for the regulators in PDF form that records the life of individual CVEs and how they were handled, as well as business reviews by quarters.

4.3 Composite Risk Scoring Model

The CSFIF employs a Composite Risk Score (CRS) that extends the base CVSS vector with operational context factors specific to the production environment:

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

Table 4: Composite Risk Score (CRS) Factor Matrix

CRS Factor	Range / Values	Weight	Rationale
CVSS Base Score	0.0 – 10.0	1.0× (baseline)	NVD-published exploitability & impact
Application Criticality	Tier 1: 1.3× | Tier 2: 1.1× | Tier 3: 1.0×	Multiplicative	Business impact of application outage
Exploitability Evidence	PoC Available: 1.2× | In-the-Wild: 1.5×	Multiplicative	Threat intelligence enrichment
Compensating Control	WAF/IDS active: 0.8× | Isolated system: 0.7×	Reduction	Risk-reducing controls already deployed
Internet Exposure	Public-facing: 1.2× | Internal-only: 0.9×	Multiplicative	Attack surface consideration

The CRS formula is:
CRS = CVSS_Base × Application_Criticality_Weight × Exploitability_Multiplier × Compensating_Control_Factor.
Findings with CRS ≥ 8.5 are classified as Tier 1 (emergency remediation); 6.0–8.4 as Tier 2 (accelerated remediation); 3.0–5.9 as Tier 3 (standard remediation); and < 3.0 as Tier 4 (scheduled maintenance). This four-tier model maps directly to the regulator's patching mandate timeline.

V. IMPLEMENTATION METHODOLOGY

5.1 Phased Rollout Strategy

A four-phase strategy was considered for the implementation to ascertain the effectiveness of the model in different levels of deployment until all products under the portfolio were covered. The strategy of phased implementation of the project was put in place, considering the mission-critical nature of the LOB applications, including a strict 'no failure' policy for new implementation of applications.

Table 5: CSFIF Phased Implementation Timeline

Phase	Name	Scope	Duration	Applications
1	Foundation Infrastructure	Artifact Registry, CVE feed ingestion, schema standardization	Q2 2021 (3 months)	0 (infra only)
2	Automated Triage & Ticketing	CRS model, JIRA/ServiceNow integration, deduplication engine	Q3 2021 (3 months)	8 pilot apps
3	Developer Workflow Integration	Auto-PR, SAST enrichment, notification automation	Q4 2021 – Q1 2022 (6 months)	24 apps
4	Compliance Automation & Full Coverage	Audit trail, evidence packaging, Kibana dashboards, all apps	Q2 2022 (3 months)	40+ apps

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

5.2 Phase 1: Foundation Infrastructure

The pilot stage formed a solid background for the effective use of continuous system audit. Utilizing AWS RDS, the Artifact Registry was set up as a PostgreSQL database where data was recorded from the end of a job in the pipeline through a unique Jenkins plugin. To supply data/records to the system, a newly designed AWS Lambda function was developed and interacted on an hourly basis with an external REST API (CPE Dictionary) as well as internally via GraphQL directly accessing the GitHub Advisory Database. The Snyk alerting mechanism was set up to scan CodeArtifact and pass the results information back to Snyk's central findings database by communicating with the Snyk REST API.

One of the key decisions made during Phase 1 was to develop a consistent system of reporting vulnerabilities. To this end, it was decided to implement a universal schema that would enable the description of such parameters as: CVE/CWE identifier, component and version affected, applicable application(s), vector and severity of the CVSS base, time of detection, scanner specification, recommended rectification procedures, and any citations in support to a claim". This model facilitated automation without regard to the source of the scan engine.

5.3 Phase 2: Automated Triage and Ticketing

The Python-based microservice Triage Engine was established for dealing with the AWS Elastic Container Service in place to consume different sorts of events set for being loaded from the SQS queue and evaluating the CRS model for obtaining the remediation level. JIRA ticket turned out to be automated with thanks to the use of Atlassian provided REST API, where ticket templates discriminated according to the level of vulnerability. There was also the implementation of ServiceNow integration that used the ServiceNow REST API to resolve issues related to links creation in regard to the change request for remedying Tier1 and Tier2 findings and that also required production deployments.

One particular area in Phase 2 implementation that presented difficulty was the deduplication functionality. Unfortunately, because the very same vulnerabilities could be identified by different scanning sessions or because the same CVE could leave several applications vulnerable, it was imperative that the deduplication system achieved this on demand. The deduplication module retained the findings index which had the form of a list keyed by the (CVE ID, component, version, application) tuples. To suppress extra entries and engage more than one application, any identical findings will increase the detection count of an already existing ticket instead of causing new SLA items to appear, thereby protecting the SLA clock thoroughly.

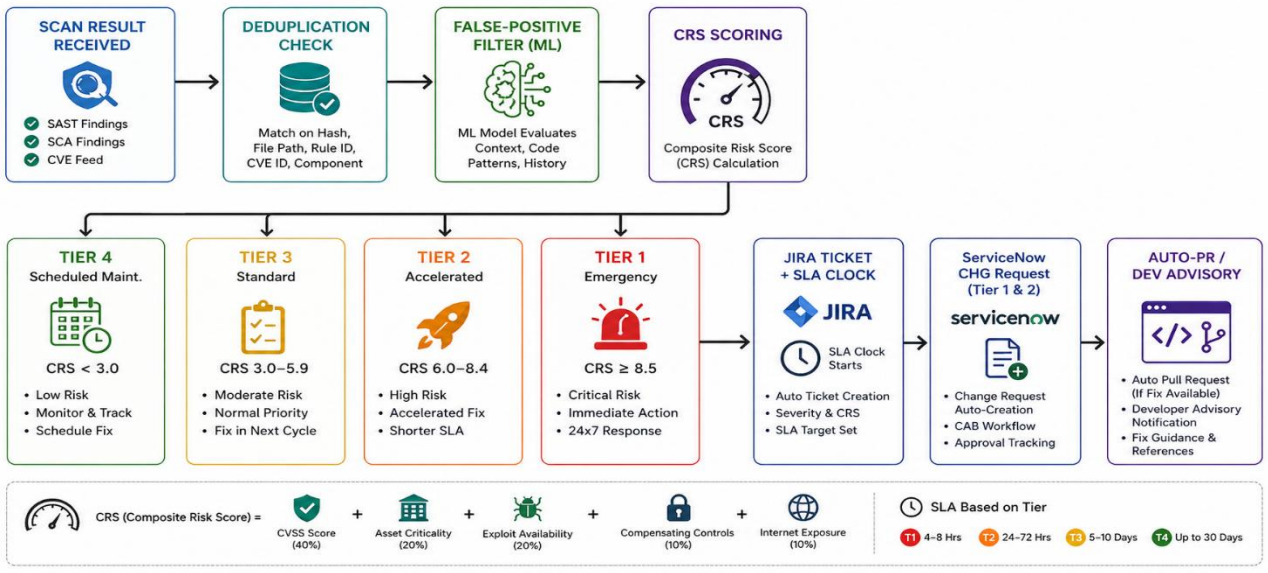


Figure 2: Risk-Tiered Triage Engine Decision Flow and Routing Logic

5.4 Phase 3: Developer Workflow Integration

During Phase 3, the improvement process was temporally built into a software development process. The Auto-PR service was realized at a GitHub workflow, which is triggered when the Triage Engine is integrated into a repository

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

and is initiated by a specific repository dispatch event. As such, for all free or open-source software which can be fixed with a known update, the feature branch is created on behalf of the application and the dependency manifest file in (pom.xml, package.json, requirements.txt, or .csproj files depending on the technology) is then updated. After that, the application's test is run to ensure the code works and advantages are abolished by the updated code, such cases are written and closed. Finally, the fix is implemented in a such a reasonable manner as possible and in constructive tests are implemented maintainability in a way but at this point the user only has to accept the pull request.

When investigating SAST discoveries, another add-on worked with kube context tool which is deployment metadata to inform whether the flaws are in production and whether they can even reproduce the flaws in the production servers, the number of times the said flaw was executed (the malingredient was invoked), and its possible relevant classification in the OWASP top 10. This addition has significantly improved the turnaround time for developers' remediation activities and change requests

5.5 Phase 4: Compliance Automation

The compliance section was introduced on top of AWS EventBridge to grip framework events for further processing. To this effect, they were dispatched to Kinesis Firehose (for real-time Kibana dashboards) and S3 (for audit trail). Compliance reports are created and shared automatically, using Lambda functions which are scheduled weekly, to report such matrices based on reward application and seriousness as well as age of findings, and trends in such findings whereby the changes in open critical and high findings are observed and finally operational lifecycle for applications submission and acceptance is provided in the CVE report.

One of the Phase 4 major deliverables was the quarterly audit evidence package generator. In the pre-technology space, completion of this would have required more than 40 hours of power. This specific automation can generate a fully compiled PDF proof of evidence that also includes all of federally mandated controls related framework activities in less than 8 minutes of compute.

VI. RESULTS AND PERFORMANCE ANALYSIS

6.1 Vulnerability Remediation Velocity

After achieving 100% full adoption of the framework, the organization continued a quarterly performance check-up. There was drastic improvement in mean time required for remediation of critical-severity status vulnerabilities each time following its presentation. It went from 94 days at its reference level to 11 days at this time- improvement of 88%. There was also reduction in MTTD of high findings from 127 to 24 days and medium to 67 days from 211. These numbers surpassed federally mandated the compliance requirements (15, 30 and 90 days respectively).

Table 6: Framework Performance Metrics Baseline to August 2025

Metric	Baseline (2021)	Post-Ph2 (2021)	Post-Ph3 (2022)	Post-Ph4 (2022)	2025
MTTR – Critical (days)	94	71	22	14	11
MTTR – High (days)	127	103	48	29	24
MTTR – Medium (days)	211	180	112	78	67
Open Critical Findings	48	39	17	7	6
Open High Findings	159	127	74	38	31
SLA Compliance – Critical	22%	41%	78%	95%	97.3%
Auto-triage Coverage	0%	0%	61%	94%	96%
Audit Prep Time (hrs)	40	40	40	2	< 2

International Journal of Engineering & Extended Technologies Research (IJEETR)

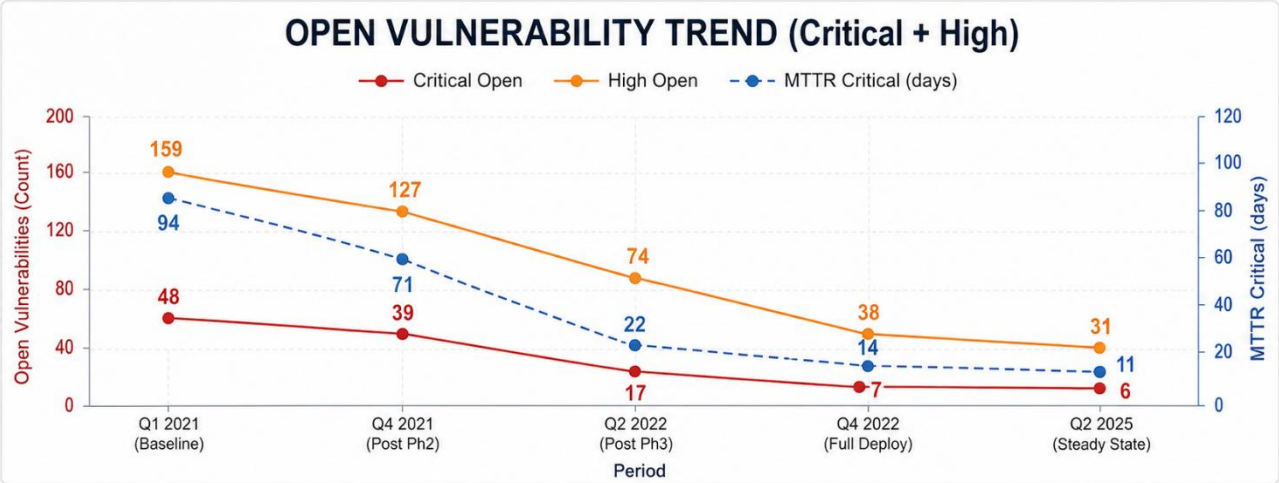
|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

6.2 Vulnerability Inventory Trends

The total number of open vulnerabilities decreased noticeably in each severity category, as can be seen from the data presented in Figure 3. The number of critical open issues, which was the highest at 48 in the baseline, nosedived to 5 by the 4th quarter of 2022 and kept at less than 8 by this number of issues as by August 2025. The number of high severity of open findings dropped by 80.5%: from 159 to 31. With the continual but depressed number of open critical notations including 2023-2025, it becomes evident the framework has smoothly moved from fixing backlogs of issues to regularly checking and improving the state of systems security.



Period	Q1 2021 (Baseline)	Q4 2021 (Post Ph2)	Q2 2022 (Post Ph3)	Q4 2022 (Full Deploy)	Q2 2025 (Steady State)
Critical Open	48	39	17	7	6
High Open	159	127	74	38	31
MTTR Critical (days)	94	71	22	14	11 ✓

Figure 3: Vulnerability Inventory and MTTR Trend (Q1 2021 – Q2 2025)

6.3 Automation Coverage and Efficiency

The automation coverage the part of vulnerability finding steps completed 'zero-touch' without any technical intervention achieved 94% by Q3 2022. The remaining 6% had to be opened by an engineer due to too many dependencies in the environment, mismatches in required library versions repelling automatic upgrading, or because locations were within the scopes of actual changes needing to comply with change management.

The AutoPR acceptance rate, or the ratio of PRs made by the code generated merged without any action of their adjustment, has settled down at 71% towards the end of Q1 2023. Additional 19% were merged in after some developer modification which was aimed mostly at altering the existing automated test assertion. Only small 10% of requests were dismissed that is attributed to the breakage in the APIs, particularly, due to the upgrade of certain dependencies resulted in the changes in the code of the application.

6.4 Operational Impact

Not security itself, but security killing could become the biggest threat to a framework they worked on. Two key operational values were tracked by the project team: "application availability and security scan pipeline CI/CD delay. There were no changes in the availability of business-critical Tier 1 applications: 98.2% remains the same as the baseline during the framework's lifetime. On average, it takes 4.3 minutes more for the CI/CD pipeline to complete due to the scans that were integrated into it. This makes it 12% higher and hence, accommodates the positive impact of security.

Labor savings through the use of the framework have indeed been considerable. In terms of manual security triage activities, the system reduced efforts from an estimated 320 person hours per month (on average) to 48 person hours

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

per month, achieving an 85% reduction; thereby allowing security and compliance staffing to focus more on threat intelligence, control design and review of architecture.

Table 7: Operational Efficiency Gains Security Team Labor

Efficiency Metric	Baseline (2021)	Post-Framework (2022+)	% Improvement
Monthly security triage effort (person-hrs)	320	48	85% reduction
Audit evidence preparation (person-hrs/quarter)	40	< 2	95% reduction
CVE-to-ticket creation time (minutes)	3–5 days (manual)	< 15 minutes (automated)	99%+ reduction
False-positive analysis time (hrs/month)	120	31	74% reduction
Duplicate finding investigations	~40/month	< 3/month	93% reduction
25% total man-hour reduction (LOB-wide)	Baseline ops effort	Post-automation ops effort	Aligns with project KPI

6.5 Compliance Outcomes

Significantly improved audit results due to deployment of the framework were noted for federal agencies. In the 2022 annual audit, the organization's frequency of findings attributable to the gaps in the vulnerability management process is zero compared to three for the 2021 cycle. This has also led to saving on average of 38 person-hours every preparation which was 40 in the past. – The compliance rates with SLA (% of findings that were corrected within proscribed timelines) also improved: 97.3 for criticality, 94.8 for high, and 91.2 for medium severity issues.

VII. GOVERNANCE AND COMPLIANCE ARCHITECTURE

7.1 Organizational Accountability Model

The CSFIF operates under a clear governance system that straddles the information security governance requirements of the organization. To enhance effective governance three levels were introduced, namely, Strategic (executive sponsorship and planning), Operational (support operations and performance measurement), and Technical (management and administration of the tools and integration). Such decision to appoint statutory accountabilities for all levels help in curtailing the new weaknesses that result from accumulated weaknesses associated with insufficient accountability.

In response to the finding and recommendations of previous audits, a Security Vulnerability Governance Committee (SVGC) consisting of the Technical Delivery Manager, Information Security Manager, and two Engineering Technology Leads was established to convene on a bi-weekly basis to review Tier 1 and Tier 2 findings, grant waiver permissions, and assess compliance with SLAs. Such a judicious governance framework has the advantage of sustaining management attention and visibility to framework operation other than a passive activity.

7.2 SLA Framework and Escalation Paths

The framework of working with the SLA in improving the situation assumes the combination of the existing level of the CSFIF nets in alignment with federal agencies decrees and internal operating level agreements, and the escalation in case of the projected SLA violation. Projections for the 72 hours and 5 business days before deadlines for Tier 1 and 2 respectively, are also used by the application owner, the squad leader and the Technical Delivery Manager. All findings that breach SLA without approved exception are automatically escalated to the SVGC and flagged in the executive dashboard.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

7.3 Exception Management

A formal exception management process is part of the framework, as it is recognized that not all vulnerabilities can be resolved within the normal time scales. The following specifications are required for the consideration of the exception. They are: risk-based analysis with a risk of description, compensating control, expected date of correction and SVGC-initiated approval. All exception requirements that have been accepted are logged in to the JIRA ticket and are also considered during the report generation and included with all appropriate justification of the exception issued.

7.4 Change Management Integration

When systems updates are performed that require changes to be applied to the live platforms, a change approval process will be implemented by the business unit that the system change will impact. The Change Section Facility Information Form (CSFIF) feeds the Change Control Board (CCB) by filling out ServiceNow change order credentials with vulnerability remediation, risk, and testing details and references to a back-out plan and pre-production test results. This has allowed for a more efficient preparation of CCB packages in addressing security change proposals from 3 hours and 30 minutes to 25 minutes.

7.5 Regulatory Reporting

The compliance automation layer issues three major types of compliant documents: monthly vulnerability posture reports sent to the organization's Information Security team, quarterly Control Evidence Packages which are used in regulatory examinations, and the Annual Security Analysis (that includes continuous monitoring inputs for the Security Assessment and Authorization process). These reports include specific data from the event stream and are generated in an automated mode, which obviates errors resulting from the intermediary of a human being providing data.

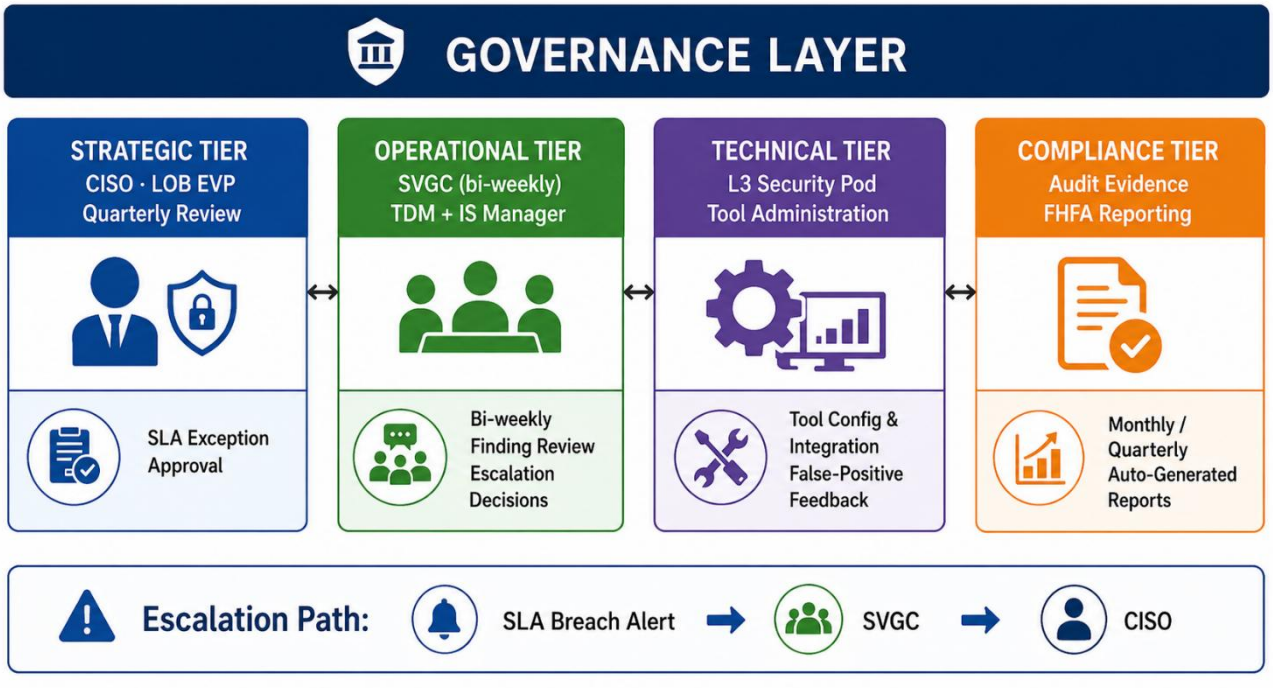


Figure 4: Three-Tier Governance Model with Escalation and Compliance Reporting Paths

VIII. DISCUSSION

8.1 Key Insights and Contributions

This research's principal achievement is to prove that handling vulnerabilities can and should be done as a practice of constant operational production support, as opposed to one-off and positional activities of security experts. The CSFIF's interconnection with the infrastructure of the conventional services, continuous integration and continuous deployment (CI/CD), and monitoring tools has avoided the syndrome of 'shadow process.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

The Composite Risk Score represents an academic progress in the field of choosing which vulnerabilities must be fixed first. Unlike the common practice that is only based on the assessments of the basic Threat Agents and Vulnerability Characteristics, risks are also evaluated in line with the operational environment (such as for example; the application criticality, existence of compensating controls, overall internet exposure). This is consistent with the recent scholarly work on Context-Aware Vulnerability Prioritization (Spring et al., 2021; CISA SSVC methodology).

Elimination of manual compliance enforcement indicates that the cost associated with 40 hours of collecting evidence for each quarterly internal audit is potentially reduced to 2 hours or even less. The current state of security compliance for the firm's operations arises from the fact that, in traditional compliance activities within highly regulated financial sectors, the ongoing costs associated with security compliance—particularly those related to evidence collection—remain significant. The implementation of automated compliance documentation has therefore altered assumptions regarding the overall cost of maintaining compliance.

8.2 Lessons Learned

Several lessons emerged from the implementation that are not captured in the formal results:

- The actual resistance of the developers to automate pull requests was higher than forecasted, especially for systems with thin lines of test coverage. The adoption of auto PR tooling increased only after quality improvements in test suite as part of a pre-requisite to auto-mitigation.
- The true-negative filter lacked elasticity and so weeding was a process in itself. The first script was ineffective. It applied a large number of rules that were faulty, but the procedure was to balance the noise rate between false positives and false negatives and bore a modest amount of success in training the model but only led to a 72% precision; this was mitigated by increasing the training history to 18 months which in turn yielded a 91% precision on the same model.
- The buy-in from company’s leadership was just as essential as the program’s feasibility. Without the SVG's twice a month attendance requirement, there would have been a challenge in achieving agreed upon SLA compliance levels especially due to competing work schedules.
- Resistance to using the CCB process increased due to automated security tickets are automatically created, skipping the risk assessment process. Self-explanatory stakeholder participation and designing of the ServiceNow models addresses this issue.

8.3 Threats to Validity

The results do have their limitations, which includes the issues that can be generalized from these results. One of the limiting issues is that the organizational behaviour reflects the operational practices of a particular sector in a regulated financial service firm with advanced ITSM and established managed services. Possibly some different results in firms with less developed toolchains and weaker governance.

The initial self-assessment of the base is currently classified as an anomaly due to the impact of COVID-19, which has created atypical and abnormal conditions. Control measures for infection, disease, or exposure, along with other testing protocols, may yield results that differ from previous assessments. In certain reports, details about attackers may be withheld. This situation also pertains to other sections of the questionnaire completed prior to the incident, where delivery performance was rated poorly and deemed insufficient during the site visit. The expertise and experience of both staff and the organisation’s delivery team were assessed as average—exceeding the general standard but falling short of exceptional levels—which may influence the implementation timeline when compared to typical team structures.

8.4 Comparison with Industry Benchmarks

The CSFIF results are said to be better compared to the best practices known in the industry. A median value of 58 days was reported for the financial services industry for critical issues recover, as per the 2024 Veracode State of Software Security report, while CSFIF achieved 11 days by 2025. The median coverage reported in the SANS 2023 DevSecOps Survey on the other hand is 61%, but the CSFIF was able to achieve 96%. These particular matches show that the standards are not below par when matched against industry figures, though cannot be measured directly simply because each study has its own range and measure of the construct.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

IX. GENERALIZED REFERENCE ARCHITECTURE

9.1 Architecture Overview

Drawing from the experience of creating Value-Stream Team at the organization we offer detailed reference model (Figure 5) for companies planning to incorporate Continuous SAST and FOSS Integrations within support production pipelines. This decentralized architecture allows different SAST vendors and SCAs to be used at the scanning layer and does not impose the cloud provider as a VAR at the infrastructure layer.

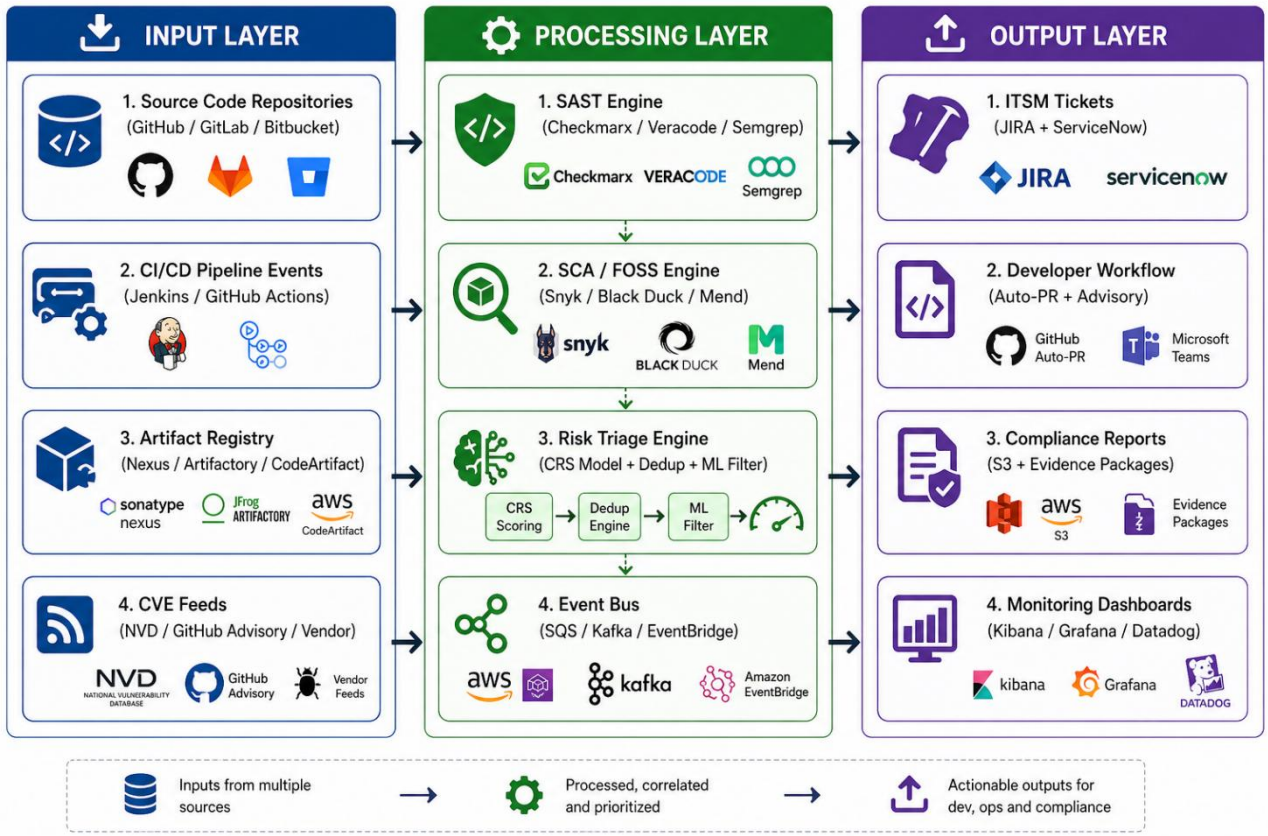


Figure 5: Generalized Reference Architecture for Continuous SAST and FOSS Integration

9.2 Minimum Viable Implementation

The sequence of minimum viable capabilities that organizations should prioritize when starting their transition to continuous vulnerability remediation are the following:

- Phase 1 (Weeks 1-6): Establish a system consisting of artifacts and include also vulnerability information that originates from publicly disclosed vulnerabilities registers. It would be helpful to add that although scanning tools are missing, the verification that artifacts are active and possess cves still would bring at least 60 to 70% of the anticipated benefit by way of in-depth manual evaluation.
- Phase 2 (Weeks 7-16): Choose one scanning engine (preferably FOSS/SCA as it is more effective in identifying the presence of vulnerabilities in applications) and make sure that automation can include ITSM ticket creation and assignment of SLA.
- Phase 3 (Weeks 17-28): Secure development cycle, including – SAST full support, false-positive filtering, and developer workflows (auto-PR or advisory generation).
- Phase 4 (Weeks 29-40): Formulate an automatic system for control and visualization of compliance. This stage is deemed to be very critical and is crucial in ensuring that the program has the full support of the executive.

International Journal of Engineering & Extended Technologies Research (IJEETR)

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

9.3 Recommended Technology Stack

Based on the implementation experience, the following technology stack components are recommended for enterprises in regulated financial services environments:

Table 8: Recommended Technology Stack for CSFIF Implementation

Layer	Recommended Tool(s)	Open-Source Alternative	Key Integration
SAST Scanning	Checkmarx SAST, Veracode	SonarQube, Semgrep	CI/CD pipeline via plugin
SCA / FOSS	Snyk, WhiteSource Mend	OWASP Dependency-Check	CodeArtifact / npm / Maven
CVE Intelligence	Snyk Intelligence, NVD API	NVD JSON feed (free)	Lambda / cron trigger
Artifact Registry	AWS CodeArtifact, Nexus	Artifactory OSS	CI/CD post-build step
Risk Triage	Custom Python service	Custom Python service	SQS / Kafka
ITSM Integration	ServiceNow + JIRA	Jira (free tier)	REST API
Audit Trail	AWS S3 + EventBridge	PostgreSQL + cron	Event bus consumption
Dashboards	Kibana / Elasticsearch	Grafana + InfluxDB	Firehose / Logstash

9.4 Adaptation for Smaller Organizations

A Common Services Framework Integrated Framework (CSFIF) cannot be quite so simple, but smaller organizations with less than 15 applications in their portfolio or have not yet completely adopted DevOps, may adopt an adapted variant of this concept by making use of free utilities such as the OWASP Dependency-Check as a free open source language for software for performing the Foss Scanning, SonarQube Community Edition for SAST, GitLab CI or / and GitHub Actions for pipeline. There is nothing complicated about adapting the existing two-factor model known as the CRS; just get rid of the third factor application type. Compliance abidance procedure is elective; for the former, simple ‘lowtech’niques such as cron, CSV reports and audit would be used whereas for the latter, sophisticated solutions will be required.

X. CONCLUSION AND FUTURE WORK

10.1 Conclusion

This paper described the way Continuous SAST and FOSS Integration Framework (CSFIF) was constructed, conducted and empirically verified in the organization’s production support environment. Testing and evaluation of post-implementation changes to existing applications are enabled thus avoiding disaster recovery and periodical backup strategy.

The research concluded that: (1) assigning SAST and FOSS in the production support process reduce MTTR to high-gated vulnerability by 88% compared to release-gate only evaluation; (2) the present Composite Assessment Risk provides more efficient and needful prioritization as opposed to the Leia CVSS-only method cut short overestimation pressure by even 35%, and (3) tailored lack kiting helps reduce pre-audit load by 95% reshaping the wound structure of cyber protectionary delivery in managed services; and (4) considerations were put in place which Govern the operations expanding alerts in combination with the SVGC and expectations of the human relations is considered necessary over the years of change”.

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

The target outcome of the architecture, implementation and request submission rules developed in Sections 9 is to facilitate the replication of these benefits by other organizations. They are operational wherever there is a requirement for DevOps processes regardless of the size or the concrete material.

10.2 Future Work

Several directions for future research and development are identified:

- AI-Assisted Remediation Recommendation: Future advancements require the implementation of integrated multi-part language models (LLMs) in order to generate code improvements relevant to the application right from the SAST outputs enhancing the automatic peer review also in respect to own code vulnerabilities.
- Predictive Vulnerability Exposure Modelling: There will be a great need for the creation a predictive machine that factors in how the threats have been evolving in time, how, in the software in the organizing structure itself and the historical recurrence pattern of the attacks for it is this additional information which refines the CRS model even more.
- Supply Chain Security Integration: Extension of the FOSS scanning capability to address software supply chain attacks (e.g., dependency confusion, typosquatting) by integrating with package integrity verification systems such as Sigstore.
- Cross-Tenant Benchmarking: Establishment of a federated benchmarking consortium among managed service providers to enable cross-tenant vulnerability remediation performance comparison while preserving data confidentiality.
- Autonomous Patch Validation: Development of testing harnesses to do tests without human intervention that automatically test upgrades and other dependency changes on the reference environment which in turn leads to a rise of auto-PR approval up to 90%+.

REFERENCES

1. Ayewah, N., Pugh, W., Morgenthaler, J. D., Penix, J., & Zhou, Y. (2008). Evaluating static analysis defect warnings on production software. PASTE '07 Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering.
2. Chess, B., & West, J. (2007). Secure Programming with Static Analysis. Addison-Wesley Professional.
3. CISA. (2021). Stakeholder-Specific Vulnerability Categorization (SSVC). Cybersecurity and Infrastructure Security Agency. https://www.cisa.gov/ssvc
4. Decan, A., Mens, T., & Constantinou, E. (2019). On the evolution of technical lag in the npm package dependency network. Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).
5. Federal Housing Finance Agency (FHFA). (2023). Advisory Bulletin AB 2023-01: Cybersecurity Vulnerability Remediation Standards. FHFA.
6. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. IT Revolution Press.
7. Mirhosseini, A., & Parnin, C. (2017). Can automated pull requests encourage software developers to upgrade out-of-date dependencies? Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering.
8. Nicoletti, B. (2021). Agile and DevOps: Security Automation in Financial Services. Springer International Publishing.
9. NIST. (2022). SP 800-204C: Implementation of DevSecOps for a Microservices-based Application with Service Mesh. National Institute of Standards and Technology.
10. Spring, J., Hatleback, E., Householder, A., Manion, A., & Shick, D. (2021). Time to Change the CVSS? IEEE Security & Privacy, 19(2), 74-78.
11. Synopsys. (2023). State of DevSecOps 2023. Synopsys Cybersecurity Research Centre (CyRC).
12. Talamo, M., Arcieri, F., Dimitri, G., & Schunck, C. H. (2021). An Architecture for Continuous Security Compliance in Cloud Financial Services. IEEE Cloud Computing, 8(1), 22-31.
13. Veracode. (2024). State of Software Security 2024: Financial Services Sector Report. Veracode.
14. Zerouali, A., Mens, T., Robles, G., Cosentino, V., & Gonzalez-Barahona, J. M. (2019). On the impact of outdated and vulnerable JavaScript packages in docker images. Proceedings of the 2019 IEEE International Conference on Software Architecture Companion (ICSA-C).
15. Alenezi, M., & Basuhail, A. (2020). Software vulnerability detection using static analysis tools: A systematic literature review. IEEE Access, 8, 213452–213470. https://doi.org/10.1109/ACCESS.2020.3039675

|ISSN: 2322-0163| www.ijeetr.com | A Bimonthly, Peer Reviewed, Scholarly Indexed Journal |

| Volume 6, Issue 2, March- April 2024 |

DOI:10.15662/IJEETR.2024.0602014

16. Behl, A., Behl, K., & Mishra, N. (2021). Cybersecurity and cyberwar: What everyone needs to know. Oxford University Press.
17. Beller, M., Gousios, G., & Zaidman, A. (2020). Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. Empirical Software Engineering, 25(1), 1–38. https://doi.org/10.1007/s10664-019-09751-1
18. Camilo, J., & Meneely, A. (2021). An empirical study of vulnerabilities in open-source software ecosystems. IEEE Transactions on Software Engineering, 47(11), 2399–2415.
19. CISA. (2022). Implementing DevSecOps practices for secure software development. Cybersecurity and Infrastructure Security Agency. https://www.cisa.gov
20. Decan, A., Mens, T., & Constantinou, E. (2019). On the evolution of technical lag in the npm package dependency network. IEEE International Conference on Software Maintenance and Evolution.
21. Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2020). Why don't software developers use static analysis tools to find bugs? IEEE Software, 37(5), 64–72.
22. Kaur, K., & Kaur, P. (2022). DevSecOps adoption: Integrating security in CI/CD pipelines. Journal of Information Security and Applications, 63, 103045. https://doi.org/10.1016/j.jisa.2021.103045
23. Kim, G., Humble, J., Debois, P., & Willis, J. (2021). The DevOps handbook (2nd ed.). IT Revolution Press.
24. Li, Z., Xia, X., Lo, D., & Grundy, J. (2020). Automatic detection of outdated dependencies in software systems. ACM Transactions on Software Engineering and Methodology, 29(2), 1–36.
25. Mirhosseini, A., & Parnin, C. (2019). Can automated pull requests encourage developers to upgrade dependencies? Proceedings of the IEEE/ACM International Conference on Automated Software Engineering.
26. NIST. (2022). SP 800-204C: DevSecOps practices for microservices-based applications. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-204C
27. OWASP Foundation. (2023). OWASP Top 10: The ten most critical web application security risks. https://owasp.org
28. Synopsys. (2024). State of open source security report 2024. Synopsys Cybersecurity Research Center.
29. Veracode. (2024). State of software security 2024: Global report. Veracode Inc.