2022

Shift Left – An Approach Note

APPLICATION SECURITY

SUNIL VARKEY

Shift-Left

The overall fabric of enterprise IT environments changed over the last few years with business and every CIO driving Application Modernization, mobility, and Cloud Adoption - critical levers of Digital Transformation.

Agility, continuous improvement, mobility, reusable components, microservices, redesigning and migration of legacy applications, API level integration federation to 3rd parties, and hybrid services with the objective of data available in 'any device' 'from anywhere' 'at any time' becoming the new norm in the application development lifecycle. The Covid era also pushed the digital transformation agenda much further in a disruptive way over the earlier years.

Exposure of enterprise applications, services, and data without adequate consideration triggered Web and applications attacks exponentially - the most significant cause of security breaches in recent years.

Few **potential reasons for this state**:

- Weak coding practices followed by developers
- Inadequate or limited testing and assessments
- Errors, Omissions, and Oversights by stakeholders
- Remediation is not always a priority due to various internal and external constraints
- Lack of ownership and accountability across technologies and platform
- Lack of mandate process around secured coding and weak production release practice
- The urgency for production release
- Application exposure to the internet (including legacy) based on architectures changes and business models
- Increased exposure attack surface area
- An easy target to identify and exploit vulnerabilities
- Reusable components – 3rd party, vulnerable codes
- False sense of security in the cloud
- Lack of control or governance on 3rd party hosted or managed environment.

Role and responsibility related to the developer, from a security perspective, changed a lot in the new agile digital transformation era – developers are the new security frontiers. But many of the developers are failing to understand their role in the security program, mainly due to

- Tight timelines, changing requirements, and too many environmental dependencies
- Security is perceived as a headwind and showstoppers from past experiences
- Repeated nonvalue-added activities due to false positives
- New or different technologies, platforms, and environment – non-standardized approach
- Usage of reusable components without any validation or auditable trails
- Lack of context on the overall picture of the criticality of the application service and its potential impact on a compromise
- False-positive mindset
- Federated service development approach, dispersed (lack) accountability
- The mindset that security is someone else's responsibility
- Security is not part of the role definition, not rewarded nor motivated
- High team turnover and workload

Traditional Security testing practices followed by many enterprises are of

- Linear sequential process approach – Waterfall model
- Standard tool-based assessment with limited manual or contextual security testing
- Security experts with NO or little application development expertise engaged in application assessments
- Assessment engagement timelines are over many weeks
- Primary security focuses on the production environment
- Standard testing approach, limited use cases based on the application service context or the existing threat landscape.
- Most of the applications in consideration are hosted and governed internally and consumed by internal stakeholders from the network.
- Limited access to 3rd party-hosted or managed environment for assessments
- The security team rarely collaborated with application developers nor provided easily consumable best practices for remediation.
- Security assessments are mainly done for compliance purpose

While most of the organization has the intent to do proper security assessments, multiple and dynamic changes in the application development lifecycle made the traditional approach inadequate, since

- Web applications are the core of digitalization, and undergo continuous changes (on average, an application goes at least three to four-time through the development CI/CD pipeline in a year which could be for bug fixes, feature releases, enhancement, etc.)
- Applications moved away from being inside layered security architecture to being directly exposed to the internet.
- Agile sprint models are not tolerant of multi-week traditional security assessment models
- Vulnerable and defective codes are getting committed to production continuously, increasing exposure and risk
- Too many development technology variants and lack of standardization – APIs, microservices, containers, workloads, different languages, web components, federated integrations, and mobile applications accessed from different devices, OS and networks.
- Microservice-based changes may not go through security change reviews
- All application components must be assessed based on the type of change and criticality considering the exposure and threats.
- Too many applications to assess based on continuous change and enhancements.
- The cost of remediation and risk is much lower in the early phases of SDLC
- Skillset, resource constraints for application security assessments
- Aligned to dynamic business changes ensuring security from all perspectives
- Repeated defects and vulnerability patterns – OWASP Top 10, limited intervention to change the pattern at the root.
- Developers coded with context may not be available for fixing if the time to assess and notify is high.
- Days and months to identify and remediate issues
- Too many applications, too many changes, and too many findings to be handled by limited resources.

The application development lifecycle must have security considerations in each of the phases.

- Functional requirement
- Systems design
- Development
- Integration and testing
- Acceptance, installation, deployment
- Maintenance

Security considerations during these phases are

- Business needs, regulatory requirements & Commitments related to security
- Security Standards and Principles
- Security control Requirements
- Secure Design solution Security Architecture
- Contextual Threat modelling
- Secure Deployment Standard
- Inventory – software bill of material
- R&R
- Remediation strategy
- Secured Lifecycle management
- Secure Dev Standards
- Logging standards
- Secure Coding Practice
- Static Code Analysis
- FOSS security
- 3rd party component security
- Vulnerability Visibility
- Secure software maintenance
- Triage of Vulnerabilities
- Criticality categorization and classification
- Threshold enforcements
- Change management – security considerations for release
- Static Code Review
- Binary Code Assessments
- Infra and Config Assessment
- Container security Assessment
- Dynamic code analysis
- IAST

These activities are time-consuming and traditionally taken up through a toll gate approach before the production release. Findings of the assessments ideally go back to the development team for review and rework. This approach worked when releases were fewer, time for testing and rework was budgeted, and there was no pressure on release timelines.

The last stage of assessment and rework is highly time-consuming, even impacting production release timeline commitments. A common practice followed by many is taking exceptions of these vulnerability risks and moving code to production – meeting compliance of security assessment, no action to fix resulting in vulnerable code in

production. Even to remediate, difficult to disrupt the service in production for remedial activities.

The cost of testing, detection, and remediation of vulnerabilities in the early stage of pre-production could be in $xx with less time, while at the last stages or in the production environment could be in $xxxx range and huge timeline, along with the associated risk.

So, application security, to be effective, practical, and sustainable security needs to integrate with the development process CI/CD pipeline or as DevSecOps, by having a set of security practices incorporated into your SDLC to build, test, and deploy secure software faster and easier.

Shift-Left

Shift left is the process to enable security considerations and assessments at the earliest stages in the development lifecycle with the participation of developers and other stakeholders.

The objectives of shift left are to ensure required security testing for the selected criteria by developers while codes are in the development and pre-production stage, better time to production deployment, reduce rework due to security findings, risk reduction, and effective use of resources.

Considerations and Candidates for Shift Left

- Applications undergoing minor changes, and bug fixes without altering security or architecture (60% of the enterprise application for testing assessment is in this category)
- Applications with low criticality
- Threat modelling by Solution architect for plotting the environment and controls in place

It is not easy to establish DevSecOps overnight, considering it a significant change involving various stakeholder groups.

- **Avoid friction between various groups and individuals, a participative approach.**

  Developers are exceptionally in high demand with a high sense of accomplishment and pride in each of the codes they develop; traditionally, they hated security groups as they were dictating terms, high time for testing, delayed the project, reported issues with ambiguous comments, non-friendly (rudeness comes when security guys don’t understand software development and its intricacy)

  Introduce security testing (on specific scenarios) as it could transform their work with less rework and better release timing. It is not easy to get developers' participation unless they understand the bigger picture and the impact they could create. Else developers will perceive this as a work delegation by the security team.

  Adequate Technical toolset and process availability

  For the selected set of shift-left activities, relevant, easily consumable toolsets should be made available in the development pipeline along with a defined actionable process to leverage at each stage correctly.

- **Automate as much as possible – self-service approach.**

  Security is not the developer's primary role, so all activities and processes should be simple. Expected activities should integrate with developers' standard routine pipeline through automation to avoid multiple routines and time-consuming manual activities. The security assessment lifecycle in the development pre-deployment phase should be delivered and consumed as a self-service approach.

- **Avoid bureaucracy and multiple toll gates.**

  By shifting left and with active participation in the security program, developers should have substantial benefits in removing bureaucratic processes and improving timelines for production release.

  With the new process, they should be able to submit the code for production release change management without getting further security or governance approvals if they execute the required tastings per the defined process. This will empower them to control the activities and their outcome.

- **Easy to adopt, consume, leverage, and mitigate**

  The primary objective of shift left is to ensure required security testing for the selected criteria by developers while codes are in the development stage. To achieve the entire objective solution, technology, the process should be easy to adopt, consume and leverage across the defined lifecycle with a clearly defined process.

Abstraction layer

One of the critical challenges with shift left adoption is educating developers on each of the testing tools (at least 5) features, configurations, and usage and frequent changes in the tool's capabilities (new and each change)

The best option to avoid this challenge is to build an easy-to-use abstraction layer as GUI for developers with minimum fields (most of the standard fields auto-populated) to input, like the application identification number, the language used, testing stage (dev phase / interim testing/production release). This layer helps to avoid the need for the developer to understand which all testing and tools are required, tool configurations, and human errors for each of the applications.

Based on this input, the abstraction application platform will be able to pull-in relevant application details from the database for the required context and define the tools and type of testing required (DAST if web component, container assessment if container used, binary testing if no codes available, etc.)

When moving from the traditional approach, we could expect some resistance from business security and 2nd line on the abstraction layer concept. The argument just for the sake of argument (they stay relevant in their role by creating complexity and confusion) will be that developers will not get to know the tools used, and they cannot leverage tools' full capability since the configuration setting is pre-defined to the defined testing policy level behind the abstraction layer.

If there are developers in the team with security expertise and are enthusiasts with extra time for advanced security testing (typically very few or nil), give them direct access to tools to do advanced testing.

Full tool native capability access to developers is not recommended since

The problem of not having an abstraction layer

- Each developer must master the features and configuration of each tool used for testing (DAST, SAST, MAST….). Very difficult to get all developers to learn every security tool.
- Can lower the level of testing configuration, ensuring mandate testing compliance is done but not effectively or comprehensively.
- Changes or introduction of (new) tools required complete re-education
- Can avoid (bypass) completing total levels of testing, ex. The only test on DAST while open-source, container codes are also in scope
- Test applications for vulnerabilities beyond their scope and control
- Service disruptions by intrusive scanning and assessments across the enterprise or on other critical applications.

Penetration testing, red team, manual testers and other specific internal security teams will be using the tools with full potential, ensuring the ROI of the investment are fully leveraged.

The abstraction layer governed by the security team should be the only application security testing toll gate considered accepted for the change management production release. This helps to ensure all applications are assessed the same way and comprehensively as defined by the security team.

Transparent

The need for each of the testing modules, approximate time for completion, pre-requisite for initiating the testing, roles, and responsibilities across the process should be well defined and transparent to all the stakeholders. If the pipeline breaks based on the vulnerability threshold (at specific points), it has to be communicated in advance, so developers will be careful to avoid mistakes.

Actionable output

Output reports from the testing should be easy to consume, relevant with minimal false positives, in-scope and out of scope of testing, and assumptions should be mentioned clearly to avoid a false sense of assurance. Required actions from the output should be mentioned with the roles of specific teams to move the process to the next phase.

Inline

The shift-left solution, tools, and abstraction layer should be on the same path as of development CI/CD pipeline, which can be called for execution easily. Any additional path or approval can consider a hindrance to adoption.

Automated decisions

The decision to evoke various testing requirements based on the application type, criticality, mandates, etc should be automated to avoid the developer's manual intervention. The system should have the capability to understand if the changes are incremental or significant, so the level of testing and time to complete testing can be improved. All relevant records, testing status, change management required pre-requisite, exception approval process all should be automated to get the full benefit of the initiative.

If change is only incremental without changing existing security or architecture, testing should be only on the changed incremental code.

Limited overhead

Shift-Left process, once in production, should work with limited overhead, reduced process completion time, and findings/exceptions recorded and updated automatically without human invention or manual errors.

Easy to Understand, Relate, Act, Fix

Security is not the primary domain of Software developers; their understanding of the security domain is limited.

Testing reports should be presented to them in an easy-to-understand and relatable mode. Reported issues should be communicated the way developers understand, with the context and risk, mentioning specific lines where the gap is identified.

The report should also provide guidelines with examples or recommended changes for the developer to fix them quickly.

Also, suggest providing a link to the internal or external training module where developers could refer to secured coding practice related to this gap.

This input could help developers avoid repeated mistakes over time and improve secured coding practices.

Extreme Shift-Left

There are situations to consider where an extreme Shift-Left approach is to be taken if too many repeated vulnerabilities will be found in later testing stages, leading to situations of time-consuming reworks, and frustrating developers.

Like spell check in Word, SAST tool with limited and specific feature capability can be integrated in the development environment, which checks codes for best practices and common errors deviations (defects) while codes are written and suggest change options to help the coder to improve coding practices while coding – helping to avoid such mistakes in future.

At each stage of testing, the error threshold (tolerance to critical, medium, low vulnerabilities) is to be set, which breaks the pipe forcing the developer to review and fix the issue before proceeding further with bad vulnerable

codes. If this can be enforced before integrating smaller modules, fixing will be much easier.

Centralized Vulnerability Dashboard

At an enterprise level, a centralized vulnerability posture storage and dashboard is required where all vulnerability ingestion, aggregation, normalization & triage happens.

Findings from various tools are aggregated and normalized against a unique value (asset name or an identifier from the software, application asset register) for standard reference and search.

This dashboard act as the authoritative source for all enterprise-known vulnerabilities in the production environment (if the build and test environment is not segregated from the production environment, then findings from those systems and environment should also be part of the consolidated dashboard), reported based on its criticality and priority.

Remediation prioritization is generally based on the criticality of the vulnerability (impact, ease of exploits), asset's criticality, service it is running, exposure of the application – internet-facing or internal, regulatory applications, hosting, or serving sensitive information, etc. This way of categorization will help the stakeholders to prioritize remediation activities.

Defects and Vulnerabilities are different; we should approach how each is to be treated and reported.

In the shift-left scenario, only findings from the last assessment before the production movement commit are to be updated in the centralized vulnerability dashboard to maintain real-time (near) known security posture.

Pipeline Assessment vs. Release Assessment

Developers require the option to assess the code at various stages of development and not only at the time of production movement stage. Initial scans may find so many common gaps, which the developer fixes on the move. These findings should not be reported to the enterprise vulnerability dashboard since those are not real production-level vulnerabilities, but these findings could pollute the overall vulnerability posture reports.

Considering the requirement, options for different modes of assessment to be provisioned in the shift-left abstraction level.

- Pipeline level assessment – all testing features, can choose the testing category required (SAST/DAST/MAST..), reports accessible to the developer but central repository or dashboard not updated with the findings
- Release assessment – All testing features available, abstraction layer logic decides testing category based on the application context, reports accessible to the developer, and updated in a central repository, all dashboards updated with finding details.

Change Management and Exception Approvals

With the Shift-Left approach, developers should be able to initiate a change management process (with all other pre-requisite completed) for moving the code to production release directly without the intervention of the security team or other process for security clearance if they completed the defined security testing through the abstraction layer, findings are within threshold and security assessment completion report with 'Pass'.

In the production movement assessment mode, the abstraction layer platform should be able to provide a consolidated report of all findings, validate open findings against the defined threshold, and provide a 'Pass' or 'Fail' report based on the defined threshold, risk appetite, which will be the pre-requisite document for the change management team to consider security findings complete and approved for production release.

If the report mentions 'Fail', application owners should prioritize fixing reported vulnerabilities to ensure open findings are within the defined threshold.

The change management process should use this report with 'Pass' as completion of required security clearance. The abstraction layer can also automatically upload this report to the change management program.

There could be situations where the business requires this application change release in the production environment immediately. Stakeholder ready to accept the risk of open vulnerabilities or compensating controls exist in the production environment due to which this open vulnerability is not a significant risk, or stakeholders need more time to mitigate the risk, willing to take the risk during that period.

The abstraction layer platform should have the feature to accept and record exceptions approval from designated stakeholders and provide a 'Pass' report based on the exception given for production release.

Authority to provide exceptions and acceptance of risk should be only for named individuals/roles based on the criticality of the vulnerabilities and their impact at application, service, and enterprise levels, including reputational, availability, confidentiality, and integrity.

All exception approval should be time-based.

Visibility, Software bill of material

Visibility across enterprise applications and its software bill of material is critical to the security program's success.

An accurate application asset register is required to ensure all applications are assessed and mitigated at any point in time for compliance and avoid blind spots.

Tools and Assessments which could be leveraged for Shift-Left

- Threat modelling (Solution architect)
- Static Application Security Testing - SAST
- Dynamic Application Security Testing – DAST
- Interactive Application Security Testing – IAST
- Mobile Application Security Testing - MAST
- Free and open-source software – FOSS
- Software Composition Analysis - SCA
- Infrastructure Security Assessments
- Container Security Assessments
- Binary Code Assessments

Challenges for Shift Left

- Maturity of the enterprise – technology, process, people
- Mindset and resistance from the team – developers, security, management
- Management support for Enforcement

Desired outcome

- Avoidance of Vulnerable defect codes moving from pre-production to production IT estate
- Sustainable Application security assessments
- The reduced vulnerable attack surface
- Effective security program reduced budget need
- Effective oversight around Pre and Post IT production environment

Empower developers as security champions

- Process governance for production movement by developers without the involvement of the security team involvement
- Participative role in the security program
- Risk acceptance process with business stakeholders on the defined guidelines.
- Trained and comfortable with adoption, assessment, report review, and remediation – Investment
- Culture with security mindset and security as a collaborative group with common business objectives
- Policies for responsible adoption and actions

Shift-Left is the participative way to ensure application security assessments are done effectively in a timely manner with the involvement of the right stakeholders and to avoid huge rework of the applications just before production release. It is no way delegation or transfer of accountability from the security team to developers. The security team is accountable and responsible to ensure testing configurations, effectiveness and process is adequate and comprehensive for the scope of Shift-Left