



ASPM Checklist

17 must-haves when evaluating, implementing, and operationalizing an application security posture management (ASPM) solution.



What is ASPM?



Application security posture management (ASPM) is revolutionizing the way teams secure their modern applications and software supply chains. Evolving from traditional application security testing tools (DAST, SAST, SCA) and the shift-left security (DevSecOps) movement, ASPM promises to maintain speed and efficiency by taking a contextual, risk-based approach.

According to Gartner, 60% of software organizations will have an ASPM by 2026.

[Read the Gartner report →](#)

But not all ASPMs are created equal.

In this checklist, we'll cover the 17 fundamental components to look for when evaluating, implementing, and operationalizing an ASPM solution to:

-  Leverage your AppSec and development resources more efficiently.
-  Proactively improve your application security posture and reduce risks.

17 must-have ASPM components



A single control plane for risks

Whether your ASPM integrates and connects findings from third-party solutions or identifies findings with built-in solutions, your ASPM must provide a unified view of application risks.

1. **AppSec tool integrations.** Your ASPM should seamlessly integrate with your AppSec testing suite to provide a holistic view of findings for correlation, prioritization, assessment, and response. Some ASPMs also map security coverage to uncover gaps where tools should be deployed.
2. **AppSec solutions.** Mature ASPMs also include native support for application security solutions such as secrets security, API security testing, software composition analysis (SCA), etc. These may come in the form of a proprietary solution or managed open source solutions. If your ASPM doesn't have built-in support for a specific solution, it should have, at the very least, a robust model for parsing that particular type of finding and enriching them with necessary insights.
3. **Automated risk assessment.** Holistically assessing application risk and providing an overall risk score is part of ASPMs' mission to benchmark and improve overall application security posture. On top of that, providing trend-based and point-in-time dashboards and reporting for AppSec KPIs such as MTTR and identified vs. fixed risks over time is a core ASPM capability.

Continuous application inventory

Contextualizing security findings for prioritization, assessment, and response requires a foundational inventory of components across layers and the software development lifecycle and the greater depth and breadth your ASPM has, the more valuable the context will be.

4. **Continuous code analysis.** Code analysis is the foundation for prioritizing security findings based on risk. By integrating with your source control manager (SCM), deep, code-based ASPMs create a complete application visibility and attack surface map that can be accessed and traversed to determine where to invest resources and answer essential questions about your application.
5. **Material code change detection.** When new application components, APIs, data flows, etc., are added or changes are made, your application attack surface may be materially impacted—introducing risk. Automating the detection of those material changes is essential for keeping up with the pace of agile development and triggering the right AppSec processes at the right time.
6. **Design insights.** Integrating with your ticketing system in the design phase allows you to analyze and pinpoint potential risky feature requests for agile threat modeling and pulling in insights.
7. **Runtime context.** Connecting with runtime sources such as API gateways and Kubernetes clusters is essential for prioritization context and getting a complete view of risk.

Multidimensional prioritization

To help teams save time triaging alert backlogs and focus on critical risks, ASPMs go beyond aggregation with risk correlation, root cause analysis, and risk-based prioritization.

8. **Basic risk factors.** Support for basic CVSS and CWSS scores and insights as to whether a finding aligns with an industry benchmark (i.e., CIS, OWASP Top Ten) or databases (i.e., CISA KEV, EPSS) are important for understanding the severity of findings.
9. **Application architecture intelligence.** Understanding where vulnerabilities are in your applications, whether deployed, internet-facing, reachable, or behind an API gateway, is required to understand how likely a finding is to manifest into a real risk. It's also essential to connect disparate findings that may result in toxic combinations.
10. **Risk impact context.** Determining whether a risk is in a high business impact application (i.e., it handles PII/PHI/PCI or is high revenue producing) is required for efficiently allocating resources to reduce risk. Additionally, impact may be increased if a risk is shared across modules, thus affecting multiple parts of your application.

Automated and proactive risk response

Once you know which risks to prioritize and have an accurate attack surface map, your ASPM should give you remediation guidance, a policy engine, automation workflows, and integrations to efficiently trigger processes and prevent risks early in the development lifecycle.

11. **Actionable remediations.** Fix suggestions, tying risks to code owners, and automated actions are all important to expedite remediation cycles between security and developers and improve mean time to remediation (MTTR).
12. **Developer workflows.** Embedding guardrails in developer tools and workflows is the only way to prevent risks from being released. Furthermore, flagging only high business impact risks prevents unnecessary blocking of releases and friction between security and development teams.
13. **Process triggering.** Not all risks necessitate a straightforward remediation. Design flaws and material code changes require further investigation. Using automated workflows and code as a source of truth—rather than self-attestation—makes triggering AppSec processes much more efficient and effective.

Bonus: Native security solutions

With supply chain security support built-in, ASPMs can natively surface pipeline and source control security weaknesses, provide better intelligence of supply chain components, and ensure complete, interconnected coverage.

14. **Software composition analysis.** Because of modern applications' overwhelming reliance on third-party components, it's crucial to have visibility into the packages and dependencies in use. Being able to natively detect vulnerabilities and license compliance issues is a major bonus.
15. **Software bill of materials (SBOM).** Being able to produce an SBOM for compliance and regulatory requirements is an important component of an ASPM and is complementary to robust application attack surface mapping. Take your SBOM further with an XBOM (eXtended bill of materials).
16. **CI/CD security.** Pipelines are the lifeblood of modern applications and are increasingly becoming targets for attackers. Maintaining visibility into all pipelines and detecting weaknesses within pipelines and pipeline dependencies is required for mapping your application attack surface.
17. **SCM security.** Similarly, your ASPM should provide insight into how source control managers (SCMs) are configured, whether branch protection rules are in place, permissions are right-sized, and malicious activity is flagged.

To protect complex modern applications while keeping pace with agile development, application security and development teams need to take a risk-based approach. ASPMs make that possible by unifying application risk assessment, visibility, prioritization, and remediation. But an ASPM is only as strong as the context it leverages.

Apiiro takes a deep, code-to-cloud approach to ASPM, combining an open platform approach with native AST and SSCS solutions to help even the most advanced AppSec teams optimize their programs for effectiveness and efficiency. Learn what sets Apiiro's ASPM apart from the rest.

[Get an Apiiro demo](#)