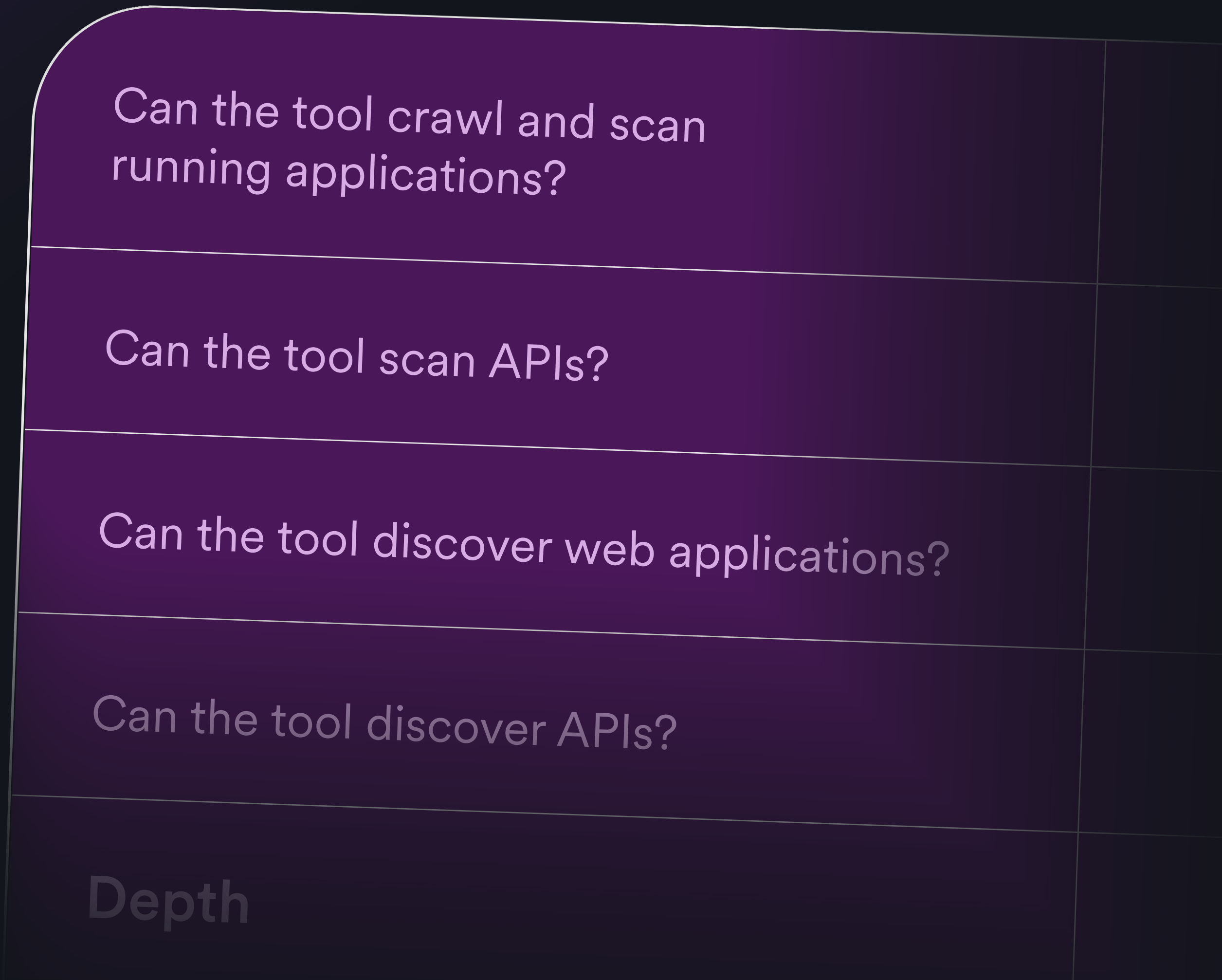
Buyer's Guide

invicti

Evaluating Application & API Security Tools

A practical checklist for modern AppSec teams



Can the tool crawl and scan running applications?
Can the tool scan APIs?
Can the tool discover web applications?
Can the tool discover APIs?
Depth

Table of contents

Introduction

Who is this guide for?

This guide is for CISOs, AppSec leaders, and security-minded engineering managers who are responsible for choosing the tools and platforms that secure their web applications and APIs. Whether you are leading a mature AppSec program or formalizing your security stack for the first time, this guide is designed to help you invest in capabilities that actually reduce risk in production, not just add more findings.

How does this guide help you?

The guide focuses on five core areas that matter most when you are evaluating application and API security tooling: coverage, depth, accuracy, automation, and risk reduction. For each area, it explains why the capability matters, what to look for in a product, and how to tell if it will work in practice for your teams and workflows.

The perspective here is deliberately DAST-first. Automated dynamic application security testing – scanning your running applications and APIs – provides a scalable way to show what is actually exploitable in production. Other signals from SAST, SCA, infrastructure, and cloud tools are still important, but they are only practically actionable when unified on a platform that uses DAST to validate, prioritize, and drive real remediation work.

What products can you evaluate?

You can use this guide to assess individual testing tools as well as broader application security platforms. It is especially useful when you are:

- Comparing dynamic application security testing (DAST) products
- Evaluating platforms that put DAST at the core but also include API security, SCA, or other testing methods
- Assessing application security posture management (ASPM) or AppSec platforms that promise a unified view of application and API risk

Throughout, the emphasis is on tools and platforms that help you find and fix real, exploitable vulnerabilities in running applications and APIs, then roll that insight up into a single, actionable view of risk across your portfolio.

Ways to use this guide

You can use the guide to structure your internal tool evaluation, identify capability gaps in your current stack, or prepare for vendor conversations. Each evaluation point is designed to map to a practical question you can ask vendors (or yourself) to separate the must-haves from the nice-to-haves for your specific needs.

At the end of this guide, you will find a checklist that brings together all of the evaluation points in one place.

Coverage

How much of your attack surface can you test?

Can the tool crawl and scan running applications?

Why you need it

A security scanner is only as good as its ability to reach all parts of your application environments. If it can't crawl dynamic pages, execute JavaScript, or follow complex navigation flows, it won't see key parts of the application, leaving these parts without security testing. With modern web applications that rely heavily on client-side transformations and rendering, failing to scan deeply means missing serious vulnerabilities.

If you're evaluating a code-focused tool like a SAST scanner, be aware that static analysis can only see what's written in the source – it won't catch issues that depend on runtime behavior, dynamic content, or client-side logic. Without scanning the running application and performing dynamic application security testing (DAST), you're missing the very part of the attack surface that real-world attackers can and will target.

Features to look for

An effective scanner should be able to handle modern JavaScript-heavy applications, including single-page apps that rely on dynamic content rendering. Look for a crawling engine that can adapt to application state, execute scripts, follow client-side routes, and interact with forms and buttons as a user would. The best tools also let you define custom crawl rules, add manual entry points, and view coverage maps that show exactly what parts of the application were reached during scanning.

How to evaluate

- Ask the vendor what JavaScript frameworks the crawler supports and how it handles SPAs or client-rendered content.
- Request a demonstration or test run against a modern web application with dynamic routing and interactive components.
- Check whether the tool provides crawl reports or coverage maps that show what was reached and tested.
- During evaluation, monitor whether the scanner can discover hidden or dynamically generated elements.

Importance

Critical

Risks of not having:

- Large parts of the application may go untested.
- Vulnerabilities may remain undetected in dynamic content.
- Code-based tools alone can't validate or cover runtime risks.

Can the tool scan APIs?

Why you need it

Modern applications rely heavily on APIs to deliver data and functionality to users – and attackers know it. APIs often expose sensitive operations like authentication, data access, and business logic, making them a prime target. If your security testing tool can't scan APIs directly, you're leaving a major part of your attack surface untested. This makes API security testing crucial across all industries and application types, especially as organizations move toward microservices, mobile-first interfaces, and third-party integrations.

SAST tools can analyze code-level issues for known API endpoints, but they won't see runtime vulnerabilities, authorization flaws, or configuration problems that only appear when the API is actually running. Dynamic testing is the only way to simulate how attackers would interact with your live APIs.

Features to look for

A capable tool should support scanning a wide range of API formats, including at least REST, SOAP, and GraphQL. It should be able to import API definitions (such as OpenAPI/Swagger or Postman collections), handle common authentication methods, and automatically test endpoints for issues like injection, broken access control, and security misconfigurations. For complex or undocumented APIs, it should also support manual entry and offer flexibility in request customization.

How to evaluate

- Ask vendors what API types and specifications (e.g., OpenAPI, Swagger, GraphQL, SOAP) their tool supports.
- Confirm whether the scanner can test undocumented or dynamically generated endpoints.
- Check if the tool supports bearer tokens, API keys, OAuth, and custom headers for authenticated scanning.
- Import an API definition during evaluation and verify that endpoints are correctly parsed, tested, and reported on.
- Observe whether the scanner uncovers real vulnerabilities without requiring excessive manual tuning.

Importance

Critical

Risks of not having

- APIs may go untested despite handling sensitive operations and data.
- Business logic vulnerabilities may be missed entirely.
- You will need a separate tool to cover APIs.

Can the tool discover web applications?

Why you need it

Before you can secure an application, you need to know it exists. In many organizations, especially large or decentralized ones, it's easy for web apps to be deployed without being formally cataloged or added to security workflows. These forgotten or shadow applications often lack proper security controls and are prime targets for attackers. Web asset discovery capabilities are especially important if you're consolidating systems, onboarding new acquisitions, or managing a sprawling digital presence.

Static testing tools and manual inventories can't identify assets that aren't already documented or tracked in source code. Dynamic discovery helps you uncover real, live web applications – including the ones nobody remembered to tell you about.

How to evaluate

- Ask the vendor how their discovery engine works and what data sources it uses.
- Request a discovery report for your organization's domain or IP range.
- Look for capabilities that allow auto-enrollment of discovered apps into scan schedules.
- Verify whether the discovery process can be customized or filtered to avoid irrelevant results.
- Test how quickly and accurately the tool surfaces assets you weren't already tracking.

Features to look for

A good tool will use multiple techniques to identify web assets, such as crawling public DNS records, analyzing SSL certificates, parsing known domains, and fingerprinting exposed technologies. It should be able to map these to your organization's known properties and surface new or unexpected assets. Bonus points if it integrates discovery directly into scanning workflows so you can go from unknown to tested in a few clicks.

Importance

High

Risks of not having

- Shadow apps may go completely untested and unprotected.
- False sense of security from testing only known assets.
- Any missed assets expand your attack surface without your knowledge.
- Untracked assets from mergers and acquisitions or decentralized teams can lead to unknown exposures.

Can the tool discover APIs?

Why you need it

An API scanning capability won't help if you don't know about an API in the first place. Just like web applications, APIs can be deployed, changed, or exposed without proper oversight, especially in modern architectures where services are built, connected, and released rapidly. Undocumented or unmanaged APIs (also called shadow APIs) are a major risk factor because they often bypass standard security checks and may expose sensitive data or functionality. If your tool can't help you with API discovery, these endpoints are unlikely to be tested or protected.

While SAST tools can reveal which APIs your code intends to expose, they only cover the code you have and can't show you what's actually running in production. Only dynamic discovery gives you visibility into real, accessible endpoints across your environment.

Features to look for

An advanced tool should be able to detect APIs passively (by observing traffic) and actively (by crawling known domains and looking for API behavior patterns). It should also correlate discovered endpoints with existing definitions to flag undocumented APIs. If your applications use large microservice architectures with centralized API management, look for integrations with your API management tools. Ideally, the tool will also integrate discovery directly with testing to automatically flag risky or unknown APIs and queue them for scanning.

How to evaluate

- Ask the vendor whether their tool can discover APIs beyond what's listed in documentation.
- Find out what techniques are used: traffic analysis, API management integrations, domain crawling, spec reconstruction, response behavior, etc.
- Request a discovery scan on a known domain and check if the tool identifies undocumented endpoints.
- Test whether discovered APIs can be added to scan targets directly.
- Evaluate if the tool highlights discrepancies between discovered APIs and existing definitions.

Importance

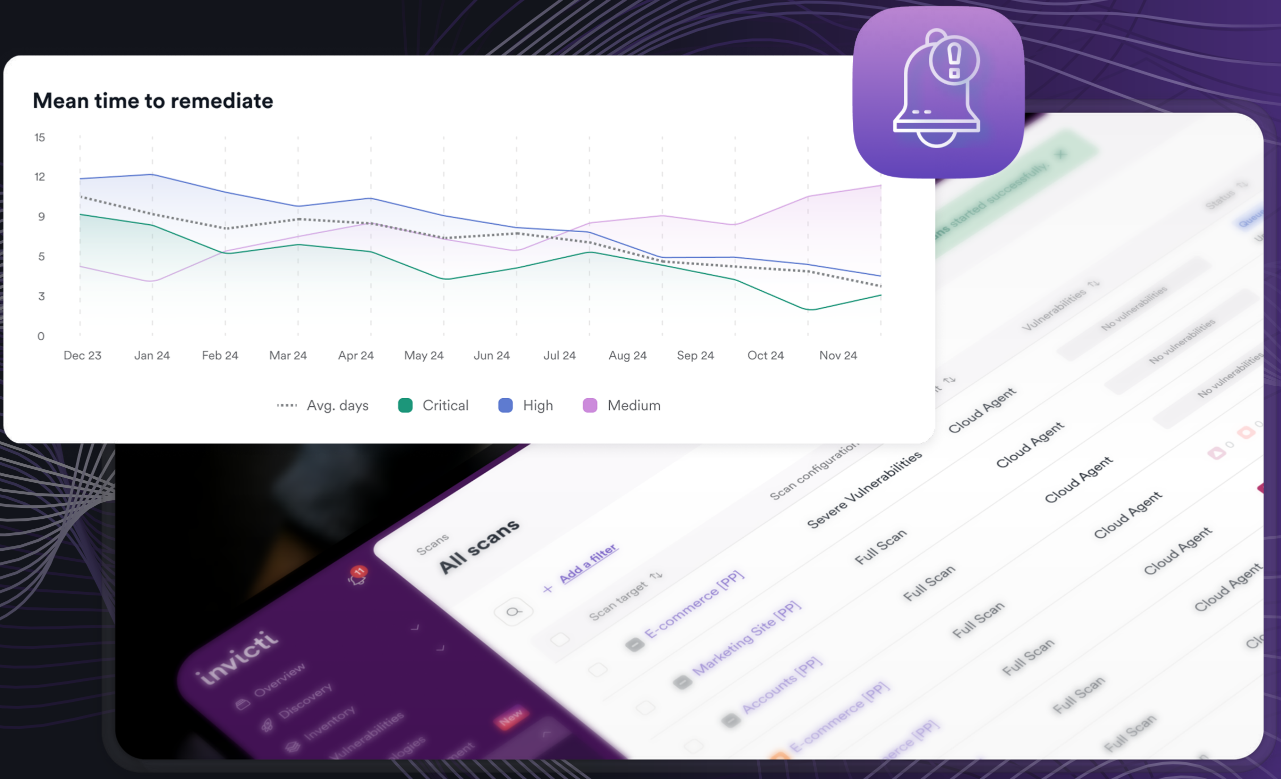
High

Risks of not having

- Shadow APIs remain untested and unsecured.
- Outdated API inventories lead to blind spots in coverage.
- Attackers can exploit APIs that aren't even on your radar.

Depth

What security issues can you find?



Can the tool find both CVEs and previously unknown vulnerabilities?

Why you need it

Application security isn't just about known threats. While CVEs represent documented and publicly disclosed issues, many real-world breaches are caused by vulnerabilities that haven't yet made it into any database – issues in custom code, misconfigurations, or flaws in business logic. If your tool can only detect known CVEs, you're likely missing the unpublicized and less predictable weaknesses that attackers still actively seek out.

Static tools, whether SAST or SCA, tend to focus on known patterns and dependencies, which means they're good at flagging typical insecure coding patterns and known CVEs in open source components. However, purely static testing inevitably falls short when it comes to detecting unknown issues that only show up in the way an application behaves when it's running. That's where dynamic testing shines.

Features to look for

Look for a tool that combines known vulnerability checks with active testing methods that simulate attacker behavior. It should test both open source components and the code behavior of your running application. The ability to generate customized payloads, analyze unexpected responses, and flag previously unseen patterns is key to uncovering unknown or zero-day-level issues.

How to evaluate

- Ask whether the tool uses active checks to go beyond CVE lookups alone.
- Request a breakdown of what types of vulnerabilities the scanner has detected in recent customer deployments.
- During testing, verify whether it identifies expected vulnerabilities that aren't tied to published CVEs.
- Confirm whether the tool can flag both dependency-level issues and runtime behavior problems.
- Look at sample scan reports to see how security issues are categorized and explained.

Importance

Critical

Risks of not having

- You may only catch what's already public knowledge.
- Custom code vulnerabilities can slip through undetected.
- Attackers can exploit business logic gaps that static tools miss.

Does the tool check for all common vulnerability types, including out-of-band issues?

Why you need it

To get a complete picture of your security posture, your testing tool must cover the full spectrum of common web vulnerabilities, also including those that don't show up in standard request-response cycles. Out-of-band (OOB) vulnerabilities, like blind XSS or server-side request forgery (SSRF), may not produce an immediate response but can still be exploited by attackers to steal data or gain access. If your tool only checks for issues that show up in direct responses, you're leaving exploitable paths open.

SAST tools, by design, are limited to what they can infer from the codebase. They may flag some issues theoretically but cannot verify whether an attack path actually works in a live environment – especially for OOB behaviors that require delayed or indirect confirmation.

Features to look for

A capable tool should test for all OWASP Top 10 categories and beyond, including injections, misconfigurations, business logic issues, and other attack types. It should support OOB testing by setting up callbacks or traps to detect whether a payload was successfully executed later. You should also be able to configure detection thresholds and enable deeper checks for higher-risk targets.

How to evaluate

- Ask the vendor if their tool supports out-of-band detection techniques and what mechanisms it uses.
- Check which vulnerability classes are covered by default and whether advanced types require custom configuration.
- During testing, include a target vulnerable to blind XSS or SSRF and see if the tool detects it.
- Review how the tool presents OOB findings and whether it provides proof of trigger or execution.
- Confirm if detection coverage extends to misconfigurations and other runtime vulnerabilities, not just code-based bugs.

Importance

Critical

Risks of not having

- Exploitable blind variants of vulnerabilities like SSRF or XSS go undetected.
- Tools may miss issues that don't show up in direct responses.
- Security teams lack confidence in coverage without OOB testing.

Can the tool identify runtime-specific security issues?

Why you need it

Many of the most dangerous vulnerabilities only appear when an application is running. These include issues tied to execution context, dynamic behavior, environment-specific configurations, and user-specific logic flows. Some components and third-party dependencies only load or execute under specific conditions at runtime, so if your tool isn't analyzing the live application, it may completely miss vulnerabilities in these dynamic dependencies.

Static tools like SAST can flag code-level risks, but they won't catch how components behave in real-world environments or how user context affects execution. You need a tool that can actively probe the application as it runs, following the same paths an attacker would to find security misconfigurations and more. Support for testing insights into the application runtime, such as lightweight IAST, can add precision in complex environments, especially when integrated seamlessly with dynamic testing.

Features to look for

The scanner should evaluate application behavior in real time, detect vulnerabilities tied to user interaction, application state, or runtime logic, and identify dynamically loaded components or code paths. Ideally, it will also highlight risks from dependencies or configurations that are only active at runtime. Built-in IAST capabilities can enhance accuracy and context-awareness without requiring deep changes to your environment (though keep in mind that some IAST tools require code instrumentation).

How to evaluate

- Ask how the tool detects context-dependent vulnerabilities such as session handling issues or inconsistent access control.
- Confirm whether the scanner identifies and tests dynamically loaded components and late-bound dependencies.
- Check how the tool handles multi-step workflows, role-based behavior, and session-specific variations.
- Test it against an application that includes conditional logic, hidden routes, or lazy-loaded modules.
- If looking at IAST, ask how it is implemented, whether it requires code instrumentation, and whether it enhances dynamic scanning with runtime insights.

Importance

High

Risks of not having

- Vulnerabilities tied to user state or logic remain invisible.
- Dynamic dependencies may go untested and unreported.
- Real-world behavior is left out of security validation.

Can the tool identify and test applications that use LLMs?

Why you need it

Software based on LLMs (large language models) is rapidly finding its way into AI features in production applications, whether through chatbots, content generation, analysis tools, or various assistants. In many cases, these modules are added with little security oversight and no consistent testing. LLM-backed applications can be vulnerable to entirely new categories of vulnerabilities, from LLM prompt injection to data leakage and indirect exploitation via integrated tools. You need to know where LLMs are exposed and whether they're introducing new attack paths.

Because LLM behavior is based on inputs, context, and unpredictable language models, these risks can't be reliably found in code or dependency scans. You need a DAST-based approach that can detect LLM usage and actively probe for real vulnerabilities. With the space evolving quickly, tools that include LLM testing will help future-proof your security program while also addressing very real risks already seen in the wild.

Features to look for

At a minimum, the tool should be able to detect when an LLM is present and exposed in the application. From there, it should attempt common LLM-specific attacks – such as prompt injection, command injection, or insecure output handling – without relying on source code access or instrumentation. The tool should also support detection and testing across any exposed entry point, not just visible chatbot interfaces.

How to evaluate

- Ask whether the tool includes dedicated security checks for LLM-specific vulnerabilities.
- Confirm that detection is based on behavior in the running application, not just static analysis.
- Include an LLM-backed feature in a test target and verify whether the tool detects it.
- Observe whether the scanner attempts relevant payloads (e.g., prompt injection, response manipulation) and reports findings clearly.
- Ask how new LLM-related attack types will be added to the tool's testing scope.

Importance

Medium

Risks of not having

- LLM-backed apps may be introduced and run without visibility or security review.
- New classes of vulnerabilities can remain untested and exploitable.
- Lack of LLM testing limits your ability to prepare for emerging threats.

Does the tool support authentication and business logic flows?

Why you need it

Many of the most serious vulnerabilities lie behind login screens or depend on specific user interactions. If your testing tool can't authenticate into the application or navigate real workflows, it won't reach the functionality that matters most: payment processing, profile management, data exports, and other logic-heavy areas. This is where real-world attacks happen, and where generic scanning often falls short.

Static tools can't see workflows at all. Even many dynamic tools stop at the login page unless manually configured. Without the ability to log in and move through your app like a user would, your tests are incomplete – and your coverage is dangerously limited. To go deeper, tools need automation features that simulate real user activity, including intelligent form filling. AI-assisted form handling can significantly boost coverage and accuracy by interacting with inputs that would otherwise require manual configuration.

Features to look for

A solid tool should support automated login for both simple and complex authentication mechanisms, including multi-step flows and token-based schemes. It should allow scripting or recording of business logic sequences to ensure deep coverage. Look for features that can automatically complete forms, choose relevant inputs, and navigate dynamic flows, making it easier to test real functionality at scale.

How to evaluate

- Ask what types of authentication are supported (form-based, SSO, OAuth, multi-factor).
- Test if the tool can maintain a session and access authenticated content consistently.
- Check if it can handle workflows that require stateful navigation, such as shopping carts or multi-step forms.
- Evaluate whether it can fill out forms intelligently without needing manual field mapping.
- Try recording a business logic flow and running a scan to see if the tool follows it correctly.
- Review whether the tool identifies different behaviors or risks for users with different access levels.

Importance

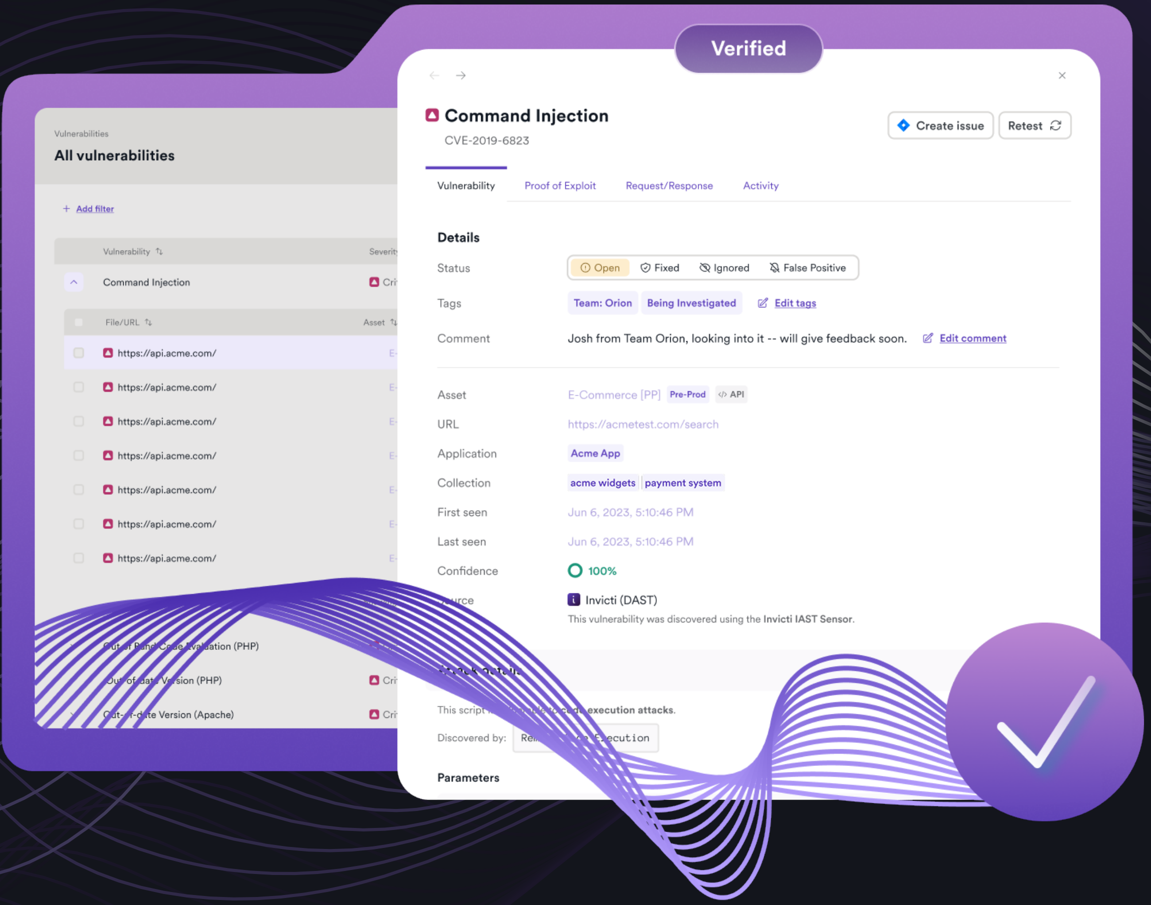
Critical

Risks of not having

- Vulnerabilities behind login screens or complex forms go completely untested.
- Business-critical workflows are excluded from security validation.
- Manual effort limits the scope and depth of automated testing.

Accuracy

How good are the results you get from the tool?



Vulnerabilities

All vulnerabilities

+ Add filter

Vulnerability	Severity
Command Injection	Cri

File/URL	Asset
https://api.acme.com/	E-
https://api.acme.com/	E-
https://api.acme.com/	E-
https://api.acme.com/	E-
https://api.acme.com/	E-
https://api.acme.com/	E-
https://api.acme.com/	E-

Vulnerability	Severity
Out of Band Code Evaluation (PHP)	
Out-of-date Version (PHP)	Cri
Out-of-date Version (Apache)	Cri

Command Injection

CVE-2019-6823

Create issue | Retest

Vulnerability | Proof of Exploit | Request/Response | Activity

Details

Status	Open · Fixed · Ignored · False Positive
Tags	Team: Orion · Being Investigated · Edit tags
Comment	Josh from Team Orion, looking into it -- will give feedback soon. Edit comment
Asset	E-Commerce [PP] · Pre-Prod · API
URL	https://acmetest.com/search
Application	Acme App
Collection	acme widgets · payment system
First seen	Jun 6, 2023, 5:10:46 PM
Last seen	Jun 6, 2023, 5:10:46 PM
Confidence	100%
Source	Invicti (DAST)

This vulnerability was discovered using the Invicti IAST Sensor.

This script is vulnerable to code execution attacks.

Discovered by: Re... Execution

Parameters

Can the tool verify if a vulnerability is exploitable?

Why you need it

Finding vulnerabilities is only useful if you can trust the results. Without confirmation of exploitability, your team is left guessing, which can lead to wasted time, missed priorities, and unnecessary friction between security and development. Verification is what separates real, actionable threats from noise.

Static analysis tools can flag potential issues but have no way of confirming whether they can actually be exploited. That forces security teams and developers to validate findings manually, introducing delays and uncertainty. A dynamic testing tool that can automatically verify exploitability gives you a major advantage: faster triage, higher confidence, and better use of limited remediation resources.

Features to look for

Look for a scanner that goes beyond pattern matching or assumptions and performs live tests to confirm vulnerabilities. This should include proof-of-exploit for critical issues, with clear, reproducible evidence that demonstrates how the vulnerability works in practice. Ideally, verification is built into the scanning engine and doesn't require a separate step or tool.

How to evaluate

- Ask how the tool distinguishes between potential vulnerabilities and exploitable ones.
- Request examples of reports for vulnerabilities that the tool can verify automatically, such as SQL injection or XSS.
- Review how verified vulnerabilities are presented in the report – are there payloads, server responses, screenshots?
- During testing, introduce a known exploitable issue and check if the tool flags and proves it.
- Ask how the tool handles edge cases or low-confidence findings – are they marked clearly, mixed in with other results, or perhaps silently dropped?

Importance

Critical

Risks of not having

- Teams waste time chasing non-exploitable issues.
- Developers lose trust in security tools.
- Real risks may get buried under noise or deprioritized.

Can the tool generate reliable results without wasting your time on false positives?

Why you need it

False positives are one of the biggest drains on AppSec efficiency. When a tool generates alerts that turn out to be incorrect or irrelevant, it erodes trust and consumes valuable time that could be spent fixing real issues. If developers stop taking reports seriously, your security program loses credibility and risk goes unmanaged.

Static tools are particularly prone to false positives because they rely on pattern matching and assumptions without knowing how the application actually behaves. A dynamic tool that tests the running application and confirms vulnerabilities in context can dramatically reduce noise, helping teams focus on what matters.

Features to look for

An accurate tool should use techniques like evidence-based scanning, context-aware detection, and runtime validation to minimize false positives. Look for features that help differentiate confirmed issues from low-confidence flags, including automated exploit testing and clear evidence. It's even better if the tool can adapt to your environment to avoid reporting known safe behaviors.

How to evaluate

- Ask for the tool's reported false positive rate (and how it defines a false positive).
- Test the tool against a known clean application with no vulnerabilities to see if it reports something anyway.
- Look at how findings are ranked and whether "confirmed" issues are backed by evidence.
- Review whether reports provide enough context to verify results without digging through logs.
- Ask what safeguards the tool uses to prevent false alerts in edge cases or noisy environments.

Importance

Critical

Risks of not having

- Developer time is wasted on chasing down non-issues.
- Security alerts are ignored due to a poor signal-to-noise ratio.
- Real vulnerabilities can be overlooked amid the noise.
- Developers lose trust in security reports and start ignoring or bypassing security testing.

Can the tool run more than one type of security testing?

Why you need it

No single testing method can cover everything. DAST excels at identifying real, exploitable vulnerabilities in running applications, but it won't catch issues in code that never makes it to production. SAST can help catch those, but it doesn't show you what is actually exploitable. Static SCA identifies known issues in open source components – but again, without runtime context. If your tool can't combine multiple perspectives, you risk blind spots and redundant tooling.

A unified solution that brings together DAST with complementary testing methods like SAST, IAST, or SCA can give you broader coverage, eliminate overlaps, and reduce the need to juggle multiple tools and workflows. As a major bonus, if DAST is used as the verification layer, it helps reduce noise across the board by checking reachability and exploitability.

Features to look for

Look for a platform that integrates dynamic testing with at least one additional method, such as IAST for runtime insights or SCA (static, dynamic, or both) for component-level risk. It should allow you to view and manage findings from multiple sources in a unified interface. Ideally, the interface and reporting should stay consistent across test types to avoid context switching.

How to evaluate

- Ask which testing types the tool supports natively and how they work together.
- Check whether findings from SAST, SCA, or IAST are aggregated or reported separately.
- Review whether DAST results are used to validate or filter findings from other tools.
- Confirm if you can enable or disable specific testing types depending on the environment.
- Look at a report that includes multiple test types and evaluate whether it enhances clarity or creates confusion.

Importance
High

Risks of not having

- You miss issues that one testing method alone can't detect.
- Tool sprawl and uncorrelated data slow down response.
- You lack a complete, prioritized view of risk.

Does the tool use a mature and proven scan engine?

Why you need it

Accuracy, depth, and reliability all depend on the quality of the underlying scan engine. An engine that's immature, underdeveloped, or powered solely by some unspecified AI tech might miss vulnerabilities, flood your reports with false positives, or break on complex modern applications. Over time, that creates friction, erodes trust, and slows down your security program. A proven engine, by contrast, can handle real-world applications at scale – giving you the consistency and confidence needed to embed security into your workflows.

A strong scan engine also reflects years of refinement in crawling, testing, and interpreting application behavior. It's not just about raw speed or coverage but how well the tool handles edge cases, modern technologies, and production-like environments without breaking, stopping dead at the first auth hurdle, or generating excessive noise.

Features to look for

You want a scan engine with a track record of successful use in enterprise environments. Look for support for modern frameworks, proven performance across complex apps, and clear evidence of ongoing maintenance and tuning. Features like intelligent crawling, robust authentication handling, and built-in verification methods are all signs of a mature foundation.

How to evaluate

- Ask how the engine works under the hood and what types of environments it supports.
- Request customer references or case studies involving large, complex applications.
- Run test scans on applications with modern tech stacks, authentication, and edge-case behaviors.
- Review how frequently the engine is updated and how it handles new vulnerability classes.
- Ask how the engine is tested and validated internally before each release.

Importance

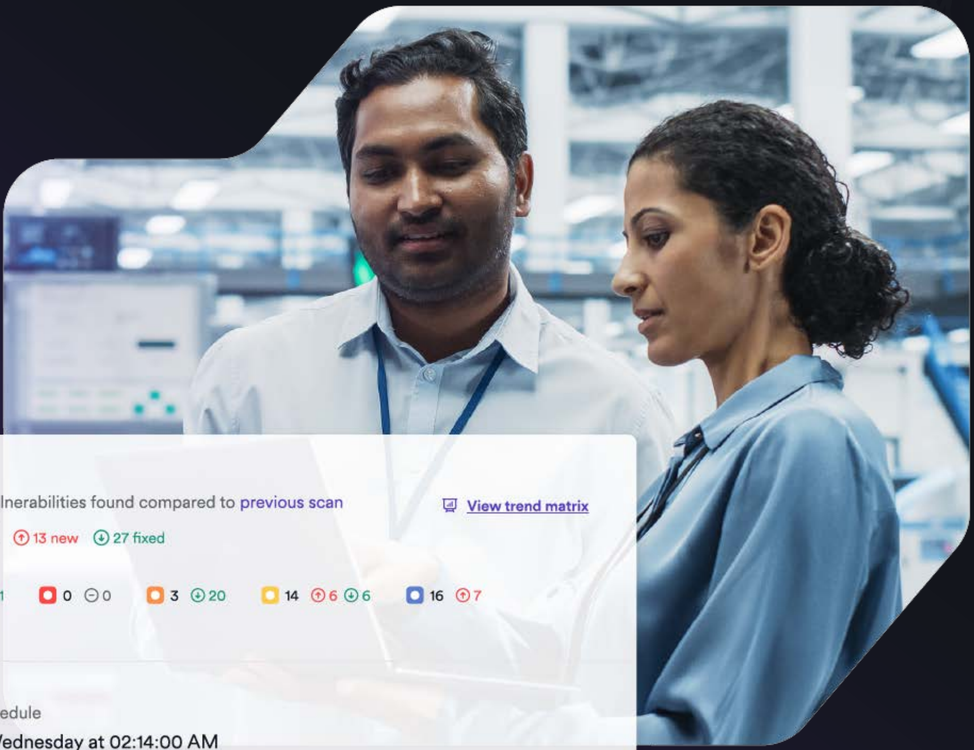
High

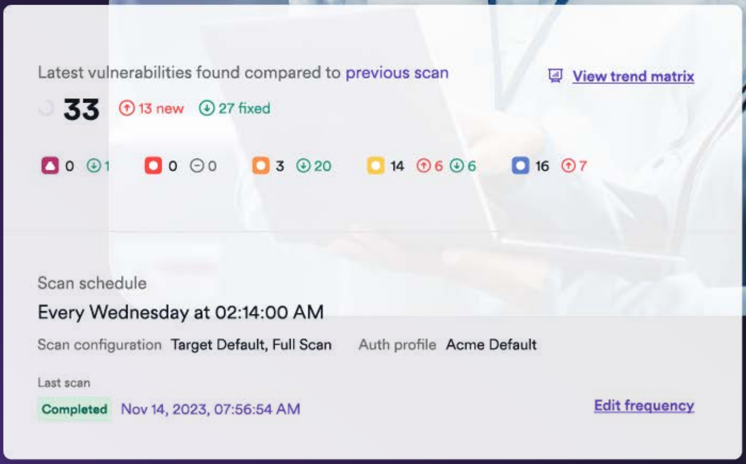
Risks of not having

- Scans may miss critical issues or report unreliable findings.
- New app technologies may break the scanner or reduce coverage.
- Tool instability undermines trust and slows down remediation.

Automation

How does the tool work with automated workflows?





Latest vulnerabilities found compared to previous scan

View trend matrix

33 ↑ 13 new ↓ 27 fixed

0 ↓ 1 0 ⊖ 0 3 ↓ 20 14 ↑ 6 ↓ 6 16 ↑ 7

Scan schedule

Every Wednesday at 02:14:00 AM

Scan configuration Target Default, Full Scan Auth profile Acme Default

Last scan

Completed Nov 14, 2023, 07:56:54 AM

Edit frequency

Can you use the tool in your SDLC?

Why you need it

Application security testing is only truly effective if it fits the way your teams build and ship software. If your tool can't run in development pipelines, support shift-left workflows, or trigger scans during CI/CD processes, it will be sidelined or run too late to prevent costly late-stage remediation. Modern AppSec requires automation because manual, after-the-fact scans just don't scale.

Static tools were built with code toolchains and CI/CD in mind, so SAST is a natural fit there. However, dynamic testing often gets left out due to outdated assumptions about speed, accuracy, or integration complexity. A modern DAST tool should prove those assumptions wrong by fitting cleanly into your SDLC without disrupting development velocity.

Features to look for

Look for native support for CI/CD integrations (like GitHub Actions, GitLab CI, Azure DevOps, or Jenkins), with the ability to run scans automatically on code merges, deployments, or releases. The tool should also provide flexible scan configurations and pass/fail logic based on custom thresholds so teams can block builds only for actionable issues.

How to evaluate

- Ask which CI/CD platforms are supported and whether integrations are built-in or custom.
- Try running a scan automatically during a pull request or deployment in a test project.
- Review how scan failures are handled: can you customize thresholds or rules?
- Check whether the tool supports scanning pre-production environments or ephemeral test apps.
- Evaluate the learning curve: can developers trigger and interpret scans without relying on security teams?

Importance

High

Risks of not having

- Security testing is delayed until after deployment.
- Teams work around the tool instead of adopting it.
- You lose the opportunity to fix issues early when it's cheaper.

Is the tool fast enough to work in dev pipelines?

Why you need it

If a security scan takes longer than your build or delays your release, developers will bypass it or turn it off completely. To be part of a modern SDLC, a testing tool must be fast enough to keep up with development pipelines, whether it's running on every commit, nightly builds, or a pre-release gate.

Speed matters because it directly affects adoption. A tool that provides the needed results quickly for a specified subset of assets can catch issues early without slowing things down. Full scans should still run on a schedule or at predefined trigger points, but pipeline integration requires performance that doesn't block progress.

Features to look for

A pipeline-friendly tool should support incremental or targeted scanning based on the scope of recent changes. It should provide tunable scan profiles and clear options to balance speed with depth. Also important are smart defaults that avoid redundant work and provide actionable results early in the process.

How to evaluate

- Ask how long typical scans take for different application sizes and environments.
- Run a scan during a test pipeline and measure the impact on build times.
- Ask about automated retesting when a vulnerability fix is submitted.
- Try configuring a partial or lightweight scan for use during active development.
- Confirm whether the tool supports scan resumption or differential scanning.
- Review how quickly initial results are available and whether they can be acted on before the scan completes.

Importance

High

Risks of not having

- Scans are disabled or ignored due to time constraints.
- Security becomes a bottleneck rather than a safeguard.
- Vulnerabilities go undetected until late in the cycle.

Is the tool accurate enough for scalable automation?

Why you need it

Automation only works if you trust the results enough to act on them without manual oversight. If your tool frequently raises false alarms or misses critical issues, you can't safely use it in a pipeline or CI/CD workflow. To scale application security, you need a tool that consistently delivers accurate, actionable findings without requiring constant human review.

Scalability means more than just performance. It means the tool can run unattended across many applications, environments, and teams without flooding systems with noise or breaking builds over questionable findings. The more confidence you have in the results, the more you can automate.

Features to look for

Look for a track record of high accuracy, especially around key vulnerability classes. Features like evidence-based scanning, exploit verification, and context-aware detection reduce false positives and build confidence. The tool should also offer flexible rules and thresholds so that only meaningful results affect pipeline decisions.

How to evaluate

- Ask how the tool ensures accuracy in automated workflows and how it avoids false positives.
- Run a scan in a test pipeline and review how many issues are flagged – and how many of those are actionable.
- Look for options to configure fail conditions based on issue severity, confidence, or type.
- Evaluate how findings are verified: are there proofs, payloads, or context included?
- Try scanning multiple projects in parallel to see how the tool handles scale and consistency.

Importance

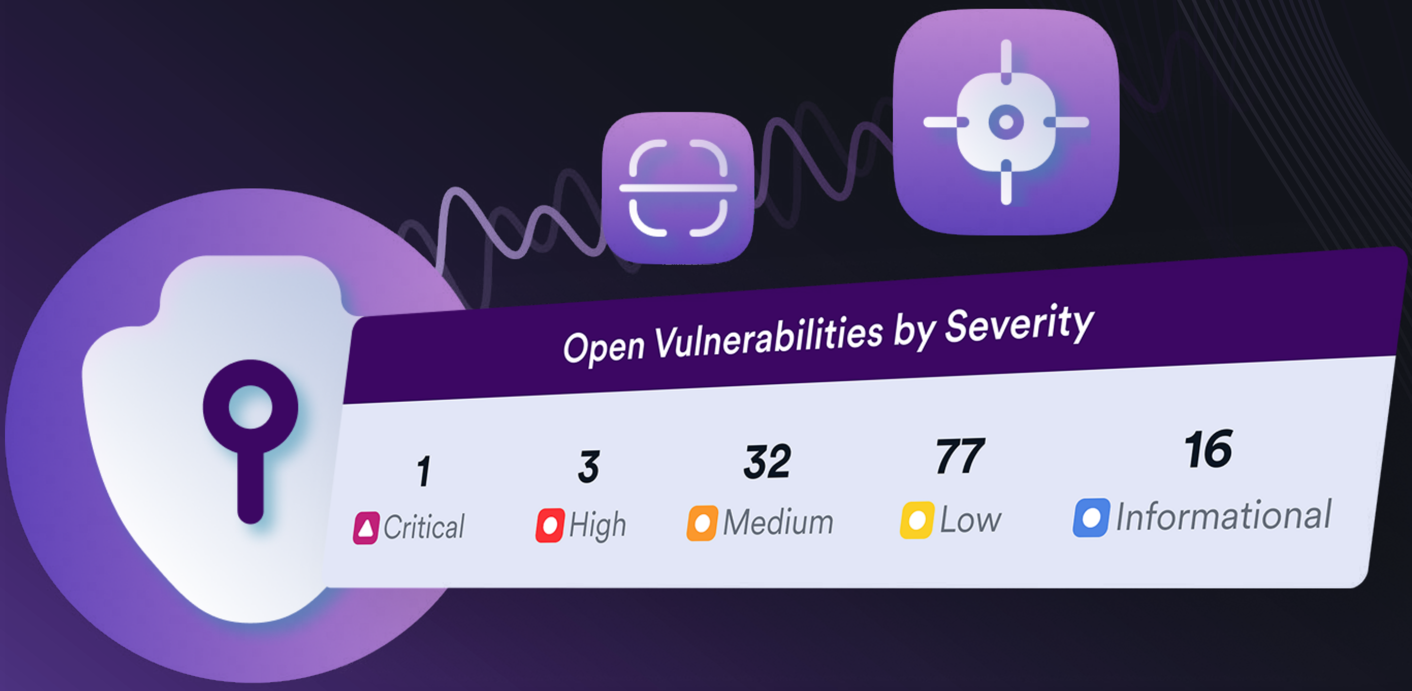
Critical

Risks of not having

- Automation breaks down under noisy or low-quality results.
- Manual triage becomes a bottleneck.
- Developers lose trust and security coverage suffers.

Risk reduction

Does the tool help you reduce actual risk?



Can the tool show you which vulnerabilities need to be fixed first?

Why you need it

Not every vulnerability is equally dangerous. If your security tool flags everything as urgent, your teams won't know where to focus – and real risks may get buried under lower-priority issues. Prioritization is essential to enable work that reduces actual risk rather than just ticking boxes.

Effective prioritization means looking beyond severity scores and labels to consider exploitability, business context, and asset value. Tools that provide this insight help security and development teams work smarter, fix the right things first, and make the most of limited time and resources.

Features to look for

Look for risk-based prioritization capabilities that go beyond CVSS scores. The tool should account for exploitability (confirmed using methods like evidence-based scanning), asset exposure, and business criticality. Bonus points for using predictive models or adaptive scoring to highlight the riskiest findings across your environment, not just the highest-rated ones.

How to evaluate

- Ask how the tool determines which issues are most urgent.
- Check whether exploitability is factored into issue ranking.
- Review whether you can tag or categorize applications by business importance.
- Test how the tool ranks vulnerabilities in high-traffic vs. low-impact apps.
- Ask whether prioritization adapts as your environment changes or new scans complete.

Importance

Critical

Risks of not having

- Teams focus on less impactful issues while bigger risks go unaddressed.
- Vulnerability overload leads to alert fatigue and inaction.
- Security efforts fail to reduce actual business risk.

Can the tool provide a risk estimate before you do any testing?

Why you need it

Running a scan gives you visibility into current vulnerabilities – but what if you could estimate risk even before testing begins? Having pre-scan risk insight helps prioritize targets, plan scan schedules, and allocate remediation resources more effectively. It's especially valuable for large environments where not everything can be scanned immediately, and not every scan result can be actioned right after testing.

While traditional tools wait for a scan to generate any insight, more advanced platforms can analyze external signals, metadata, or historical trends to estimate which applications or APIs are likely to be riskier. That gives security teams a head start on managing exposure.

Features to look for

Look for a platform that can assign preliminary risk scores to assets based on characteristics like exposure, technology stack, age, and past vulnerability history. These estimates should be visible in dashboards and usable for prioritizing discovery, scanning, and remediation planning.

How to evaluate

- Ask whether the tool provides any predictive risk scoring or similar insights before scans are run.
- Review how pre-scan risk levels are calculated. What signals or heuristics are used?
- Check whether some unscanned assets are already flagged as potentially high risk.
- Test whether any provided risk estimates help you plan which targets to scan first.
- Ask how pre-scan risk insights are updated over time or integrated into broader reporting.

Importance

Medium

Risks of not having

- High-risk apps may be overlooked during planning.
- Scan schedules may not align with actual risk.
- Resources are spent on fixing low-risk targets while bigger risks remain unaddressed.

Does the tool provide centralized visibility for security testing data?

Why you need it

Without a central view of your security posture, it's easy to miss patterns, overlook risk, or duplicate efforts. If testing data is scattered across tools, teams, or formats, you can't track progress, measure coverage, or make informed decisions. Centralized visibility turns raw scan results into actionable insights that security leaders can use to guide priorities and demonstrate accountability.

This is where application security posture management (ASPM) concepts and features really start to matter. A platform that brings together data from across your applications, environments, and teams while anchoring that data in reliable, built-in testing gives you both visibility and control. Such a testing-first approach is distinct from "pure" ASPM solutions that only aggregate data from connected tools because it ensures that what you're seeing reflects actual risk, not just metadata.

Features to look for

Look for a platform that consolidates findings from DAST, SAST, SCA, and other testing tools into a single, unified view. It should support filtering by business context, tagging by team or risk category, and tracking of vulnerability trends over time. Built-in dashboards and reporting should allow for both operational and executive-level insights without requiring separate tools or exports.

How to evaluate

- Ask whether the tool offers a centralized dashboard that spans all testing and asset data.
- Review how assets, scans, and findings are grouped, e.g., by application, team, or business unit.
- Check if findings can be filtered, tagged, and tracked across time and teams.
- Alongside centralized vulnerability management, ensure you are also getting solid security testing tech in the box (and vice versa).
- Test how the platform surfaces high-risk areas or applications at a glance.
- Ask whether visibility includes both raw scan results and contextualized risk views.

Importance

High

Risks of not having

- No way to track risk reduction or program coverage.
- Stakeholders lack visibility into what's being secured and why.
- Insights are disconnected from real testing data.

Can the tool generate reports to support compliance efforts?

Why you need it

Even though your priority should always be real risk reduction, compliance still matters. Whether it's SOC 2, PCI DSS, HIPAA, or internal policy enforcement, you need to demonstrate that applications are being tested regularly, vulnerabilities are managed, and security posture is improving. Without clear, reliable reporting, audits become a scramble – and leadership lacks visibility.

Security testing tools should support compliance not just by ticking boxes but by showing evidence of real, ongoing efforts. The best tools generate reports that map findings to compliance requirements and give auditors, executives, and stakeholders the assurance they need without extra overhead for your team.

Features to look for

Look for built-in compliance report templates aligned to common standards, with the ability to filter by application, timeframe, or remediation status. Reports should be exportable, shareable, and easy to generate on demand. Ideally, they should include proof of testing activity, vulnerability trends, and confirmation of issue resolution.

How to evaluate

- Ask whether the tool includes report templates for specific frameworks and industry standards (for example, PCI DSS, OWASP Top 10, HIPAA).
- Generate a report based on a recent scan and review how findings are presented.
- Check if reports can include remediation history, scan frequency, and compliance status.
- Confirm whether reports can be scheduled or automated for recurring audits.
- Review whether reports are understandable to non-technical stakeholders and auditors.

Importance

Medium

Risks of not having

- Audit preparation becomes manual, time-consuming, and error-prone.
- Gaps in documentation raise red flags during compliance reviews.
- Stakeholders lack proof that security work is being done.

Summary: How to get the best out of this guide and checklist

You've seen what to look for in a capable, scalable, and risk-focused application security tool. Now it's time to use that insight to guide your evaluation process.

Define what you're looking for

Start by identifying your highest-priority needs, whether that's expanding test coverage, reducing noise, integrating into pipelines, or preparing for audits. Then use the checklist to focus on capabilities that will deliver immediate impact without locking you into a narrow toolset.

Think about tomorrow's needs

Finally, keep the future in mind. Application security isn't static. The right product should not only solve your current problems but also support your roadmap, whether for tighter DevSecOps integration, better visibility, tool consolidation, or improved automation. And be sure to ask the vendor about their product roadmap to make sure it exists and aligns with your own direction.

Trust the checklist

Use the checklist that follows to run a structured, confident evaluation – and choose a solution that works for your team, not just your budget.

Ask pointed questions

As you evaluate vendors and solutions, don't just compare features. Ask practical questions that will most affect your day-to-day operations:

- How does the tool integrate into our existing workflows?
- Can the tool scale with my team and organization?
- How much manual work on triage and verification will be needed for the results?
- Will the tool help us find, prioritize, and fix the issues that matter most?
- How long will it take to go from deployment to meaningful results and value?
- What industry experience and product track record does the vendor have with application security?
- Does the vendor offer onboarding or do we need to figure out everything ourselves?

Tools that look similar on paper often behave very differently in practice, so request live demos, test real apps during proof-of-concept trials, and look for evidence that the tool and vendor can deliver on their promises.

AppSec Buyer's Checklist

	Yes	No	Partly	Notes
Coverage				
Can the tool crawl and scan running applications?				
Can the tool scan APIs?				
Can the tool discover web applications?				
Can the tool discover APIs?				
Depth				
Can the tool find both CVEs and previously unknown vulnerabilities?				
Does the tool check for all common vulnerability types, including out-of-band issues?				
Can the tool identify runtime-specific security issues?				
Can the tool identify and test applications that use LLMs?				
Does the tool support authentication and business logic flows?				

	Yes	No	Partly	Notes
Accuracy				
Can the tool verify if a vulnerability is exploitable?				
Can the tool generate reliable results without wasting your time on false positives?				
Can the tool run more than one type of security testing?				
Does the tool use a mature and proven scan engine?				
Automation				
Can you use the tool in your SDLC?				
Is the tool fast enough to work in dev pipelines?				
Is the tool accurate enough for scalable automation?				
Risk reduction				
Can the tool show you which vulnerabilities need to be fixed first?				
Can the tool provide a risk estimate before you do any testing?				
Does the tool provide centralized visibility for security testing data?				
Can the tool generate reports to support compliance efforts?				

Invicti Security provides a centralized application security platform that helps organizations prove and reduce real application risk with zero noise. Combining application security posture management (ASPM) with discovery and scanning, Invicti gives security and development teams a single, correlated view of exploitable vulnerabilities across their application frontends and APIs. With its best-of-breed dynamic application security testing (DAST) acting as the fact-checker, Invicti integrates into CI/CD pipelines to deliver proof-based results supported by AI-powered prioritization and remediation. Built on more than 20 years of DAST innovation through Acunetix and Netsparker and further strengthened by the acquisition of Kondukto ASPM, Invicti operates globally across more than 11 countries and serves over 4,000 customer organizations.

Acunetix & Netsparker & kondukto

ARE NOW

invicti

Invicti's solutions automate application vulnerability identification, confirmation, and management to help keep sensitive information and critical infrastructure secure.

Find Us LinkedIn X (Twitter) Facebook Instagram invicti.com Contact Us