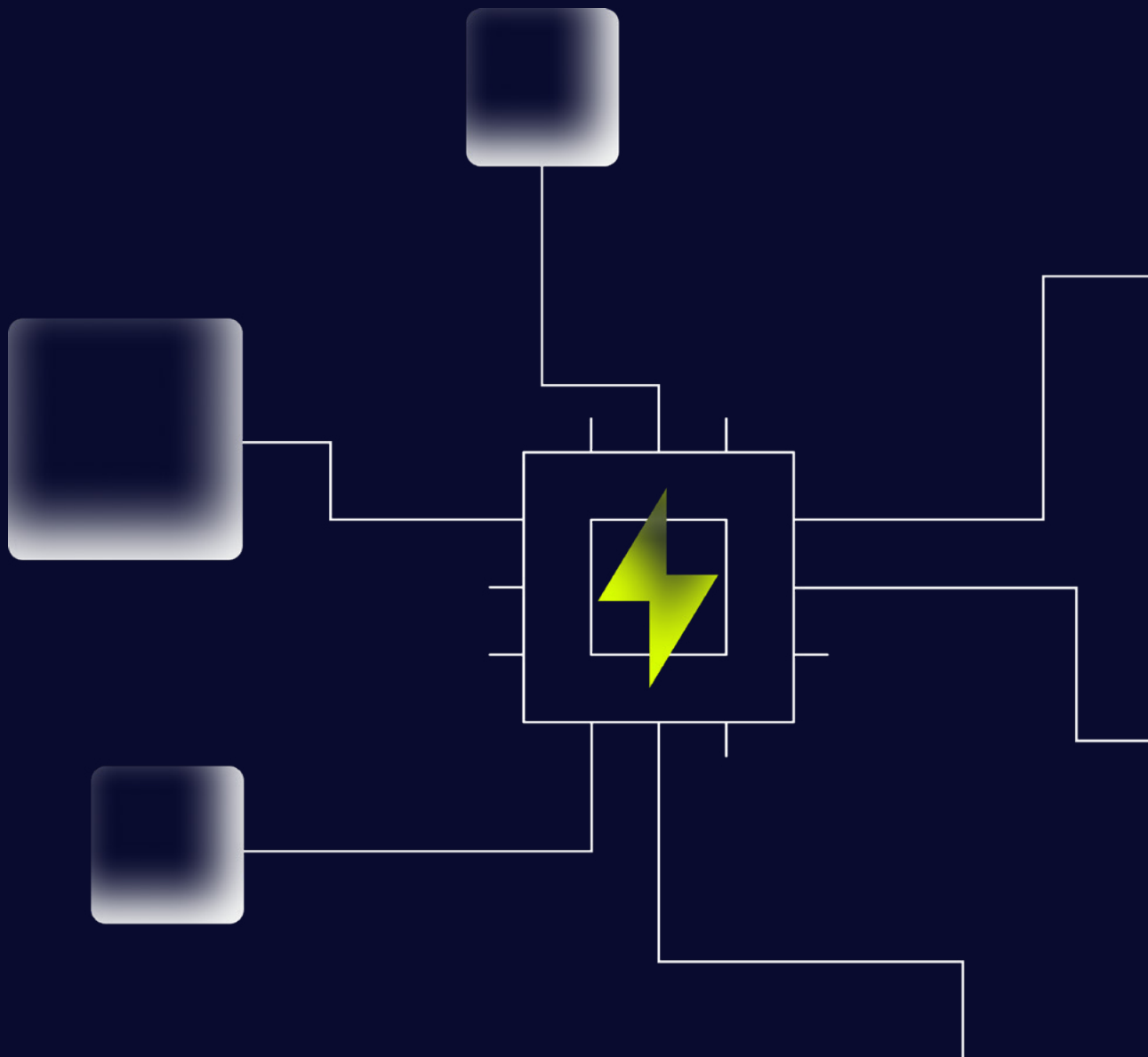
sonatype

A Buyer's Guide to Repository Security

How to Protect the Point Where Software Enters Development



A single bad package can create a surprising amount of work. What starts as a routine dependency download can quickly become a security investigation, a compromised build, or a release that cannot move forward.

Developers have to figure out where the package was used. Security teams need to understand what it did. Applications may need to be rebuilt and retested. And everyone loses time dealing with a problem that could have been stopped before development even began.

Now consider how software gets built today. Developers aren't the only ones bringing third-party components into applications. AI agents are increasingly making those requests automatically, pulling packages and other dependencies into development with little or no human involvement.

The faster software enters development, the more important it becomes to know what you're letting in. Yet many organizations still rely heavily on security checks that happen after components have already been downloaded, incorporated into builds, or distributed across development environments. Those checks are important, but by then, some of the damage may already be done.

THE FASTER SOFTWARE ENTERS DEVELOPMENT, THE MORE IMPORTANT IT BECOMES TO KNOW WHAT YOU'RE LETTING IN.

Repository security gives organizations a way to evaluate component requests before software is downloaded and added to builds. But having security as part of your artifact repository doesn't mean you're fully protected.

Buyers need to understand:

- The point at which the check happens.
- The intelligence behind the decision.
- How the control fits their repository and development workflows.

This whitepaper offers an evaluation framework for engineering, platform, AppSec, DevSecOps, and repository teams. You'll learn how to assess the risks, evaluate security intelligence, and compare repository protection approaches.

TL;DR

Creating a Control Point



Developers, pipelines, and AI agents request software components at increasing speed, making the repository path a practical place to evaluate risk.

Assess Protection Timing



A key distinction is when a solution acts, since blocking malicious components before download can prevent them from entering development, while scanning later helps identify risk in components already in use.

Configurable Policies



Buyers should assess when a solution makes its decision, what intelligence supports it, and if configurable policies can automatically block, quarantine, or allow component requests according to organizational risk.

Proving Value



A proof of value should test the same packages and workflows across vendors, then measure decision quality, timing, coverage, and operational impact.

The Risks That Come With Uncontrolled Dependency Downloads

A malicious package doesn't have to make it into production to cause problems. Some packages execute code during installation. Others may attempt to steal credentials, access development environments, or interfere with builds.

Even when a suspicious component turns out to be harmless, investigating it after it's been distributed can consume time across multiple teams. Buyers should consider the following risks and how they could affect delivery:

- **Malicious packages:** Attackers can use typosquatting, brandjacking, compromised maintainer accounts, or hijacked releases to distribute harmful code through familiar package ecosystems.
- **Vulnerable or unwanted components:** A component may contain a known flaw, violate an organization's license or security policy, or fall outside approved usage.
- **Dependency sprawl:** Teams may use different registries, repository configurations, and approval rules, making it harder to apply consistent controls or understand what entered a build.
- **Delayed detection:** If a problem is found only after a package has been used, teams may need to identify affected applications, remove or replace components, rebuild, retest, and document the response.
- **Unreviewed automated requests:** Developers and AI agents can introduce dependencies quickly. Security teams may not be able to review every request manually while still supporting delivery goals.

A security tool that identifies a problem after a component has already been downloaded isn't providing the same protection as one that stops it at the point of entry. Both approaches have a role. But if the goal is to prevent unnecessary exposure and rework, buyers need to understand exactly when a security decision happens.

The Repository's Role in Modern Development

Most development teams don't spend much time thinking about where their dependencies come from. They add a package, run a build, and expect the software they need to be available.

That's exactly how it should work. Developers shouldn't have to investigate every package before they can get something done. But someone needs to make sure the software entering development meets the organization's requirements.

A repository provides a central place to manage that process. It proxies, stores, and distributes software components, containers, models, and build outputs. Instead of every team retrieving software independently, developers and agents can access components through a common source.

WHEN A DEVELOPER OR AI AGENT REQUESTS A COMPONENT, THE REPOSITORY BECOMES A PLACE TO EVALUATE WHETHER THE SOFTWARE SHOULD BE ALLOWED INTO THE ENVIRONMENT.

This creates an important opportunity. When a developer or AI agent requests a component, the repository becomes a place to evaluate whether the software should be allowed into the environment.

Approach	Where It Acts	Buyer considerations
Security integrated with a repository	Assesses requests directly in existing repository workflows. The exact evaluation and delivery sequence depends on the implementation.	Does it evaluate requests prior to serving components? Which policies, formats, and deployment models are included?
Security gateway for an existing repository manager	Routes package requests through a separate security service alongside the organization's repository manager. It may evaluate before upstream fetch or before downstream delivery, depending on the design.	Does it block before upstream fetch or downstream delivery? Which package types and request paths are supported? What services, network changes, or licenses are required?
Application or dependency scanning	Evaluates declared project or application components to help teams identify and mitigate risk.	Does it prevent downloads or report on components post-entry? Can policies act automatically, and when?

Three Approaches to Repository Protection

Organizations can protect component use through several approaches. These can complement one another, but they do not make the same decision at the same point.

To compare options, separate your preferred deployment model from decision timing. Repository-integrated controls and separate gateways can both make request-time decisions, while differing in architecture and operating requirements.

Project or dependency scanning assesses declared or resolved components in project or build context, with enforcement depending on how it is integrated.

The right choice depends on where software enters the environment, which controls are already in place, and how much operational change the organization can support.

What Effective Repository Security Should Do

Finding malicious software is useful, but preventing developers from downloading it is better.

Effective repository security should make component decisions before they create downstream exposure. It also needs threat analysis that reflects current package activity and uses context beyond published vulnerability records. Without it, a solution may miss harmful packages or flag legitimate ones.

Buyers should evaluate when a solution analyzes a request and what intelligence informs its decision. A request-time check backed by broad, current, well-researched intelligence can help teams stop malicious components before download while reducing unnecessary blocks and investigations. Two capabilities deserve particular attention: blocking malware before download and policy-based automation.

BLOCK MALWARE BEFORE DOWNLOAD

A scan performed after a component has been downloaded or stored may help teams identify exposure, but it cannot prevent that initial download. Request-time protection evaluates a package when a developer or build requests it and can block a component before it is fetched or served downstream. This gives the organization a chance to stop known malicious or suspicious components before they reach development environments or builds.

When evaluating repository security solutions, buyers should ask vendors to demonstrate where this decision occurs:

- Does the solution evaluate the request before the package is fetched from upstream?
- Does it prevent the package from being cached or served?
- Which request paths and ecosystems does it cover?
- What intelligence informs the decision, how is it kept current, and how are uncertain findings validated?

AUTOMATE DECISIONS WITH POLICY

Policy-based automation is only as useful as the intelligence behind the decision. If that intelligence is incomplete, outdated, or lacks context, automation cannot consistently block safe components or allow malicious ones. Research and data quality are therefore essential to making policy effective.

RESEARCH AND DATA QUALITY ARE ESSENTIAL TO MAKING AUTOMATED POLICY ENFORCEMENT EFFECTIVE.

Sonatype's open source intelligence combines package-consumption signals with security research to identify and investigate suspicious components. That research gives organizations a stronger basis for deciding how policies should respond to component requests.

Policy-based automation makes decisions repeatable across requests and teams. It can reduce reliance on manual review for routine cases while giving security teams a defined process for uncertain results and exceptions. Buyers should confirm that policies can be scoped appropriately, decisions are visible to administrators, and developers receive useful information when a request is blocked.

Look for a solution that can:

- Check components as they are requested, before they reach developers or builds.
- Block or quarantine components identified as malicious or suspicious.
- Apply configurable policies to component requests.
- Explain why a component was blocked, quarantined, or allowed.
- Cover the ecosystems, repository managers, and request paths the organization uses.
- Provide visibility into decisions, exceptions, and policy outcomes.

How Sonatype Firewall Protects Artifact Repositories

Sonatype Firewall helps organizations put these principles into practice by evaluating component requests as software enters development. It uses Sonatype's open source intelligence to identify malicious and suspicious components, then applies organizational policy to determine whether or not a request should be blocked, quarantined for review, or allowed.

That decision point matters. When a developer, build, or AI agent requests a package through a protected repository path, Sonatype Firewall can assess it before it is delivered downstream. A component identified as malicious can be stopped before it reaches a developer's environment or becomes part of a build. For findings that need review, teams can use quarantine and policy controls instead of relying on every developer to investigate packages on their own.

SONATYPE FIREWALL ASSESSES AND BLOCKS MALICIOUS COMPONENTS BEFORE THE ENTER THE REPOSITORY.

Sonatype Firewall is designed to fit into repository workflows, so teams can apply consistent controls to component requests while supporting the tools and ways of working developers already use. This gives security teams a practical place to govern software intake and gives developers and agents a clearer path to the components they need.

The Intelligence Behind a Security Decision

The effectiveness of any repository security solution ultimately depends on the quality of the intelligence and analysis driving its decisions. Published vulnerability records help, but they don't tell the whole story. A malicious package may have no advisory, a newly published component may have little history, and harmful code may be hidden in a package or beyond its declared dependencies.

Database size alone cannot show whether or not a solution will recognize those risks. Buyers should look at what data and analysis inform a decision, how uncertain findings are investigated, and whether the resulting intelligence can guide action at the point a component is requested.

HOW SONATYPE APPROACHES SECURITY INTELLIGENCE

Sonatype's approach starts with the software teams actually consume. Our intelligence research draws on package-consumption data, including dependency update patterns across more than 1.5 trillion Maven Central requests, assessments of hundreds of thousands of enterprise applications, observations of malicious packages across several ecosystems, and packages blocked by Sonatype Firewall.

Machine-learning signals and AI-powered analysis help surface suspicious activity. Sonatype's security researchers investigate those findings by examining release behavior, package contents, and payloads. They can also trace code beyond declared dependencies to find related packages that package metadata alone may not reveal. In the November 2026 PolinRider attack, Sonatype researchers linked concealed loaders across 12 npm releases to the same wallet and payload retrieval infrastructure.

These sources and methods work together. Analysis can surface a lead; research helps establish what it means; the resulting intelligence informs Firewall's decisions to block, quarantine, or allow a request. Buyers should ask how each part of that process works and what evidence supports a decision.

WHAT TO EVALUATE

When evaluating repository security solutions, it's important to understand the breadth and depth of the intelligence behind the solution. We recommend looking at five dimensions.

Criteria	Questions to Ask	Sonatype's Approach
Coverage	Which ecosystems, versions, and component types are covered? How are new or less common packages handled? What telemetry, public advisories, research, package observations, and consumption patterns contribute to your security intelligence?	Sonatype has analyzed and cataloged more than 270 million open source components. Its intelligence also draws on signals from millions of Nexus Repository instances and over a trillion requests to Maven Central, giving its researchers a broad view of open source use.
Depth	Can the solution identify malicious behavior beyond published vulnerability advisories? How are suspicious findings reviewed?	Machine learning and AI help surface vulnerabilities and suspicious behavior. Sonatype researchers investigate findings, validate vulnerability details, and research threats beyond public advisories.
Context	Can the provider connect package data with how components are published, updated, requested, and used?	Sonatype's secondary expansion can connect a known vulnerability to additional affected components. Advanced Binary Fingerprinting can identify embedded, renamed, or modified components that package manifests may miss.
Freshness	How are new threats identified, reviewed, and reflected in decisions? How are false positives and research corrections handled?	Sonatype continuously monitors open source repositories for new vulnerabilities and suspicious packages. Researchers investigate and validate findings, updating Sonatype's intelligence as information is confirmed.
Actionability	Does the solution evaluate a package request before download? Can it prevent the package from being fetched, cached, or served downstream? Can developers see why a request was stopped and what they can do next?	Sonatype Firewall applies intelligence to component requests and can block, quarantine, or allow them into your repository instance taking advantage of our advanced policy engine.

Use these questions to compare the evidence each provider can substantiate. Ask what its intelligence covers, how findings are validated, and how risk information informs a decision. Then map those capabilities to your ecosystems and repository paths.

Some solutions focus on pre-download decisions at a repository boundary. Others emphasize application or dependency analysis after components are declared or consumed. Some vendors offer both. Compare equivalent capabilities and confirm which solutions, subscriptions, or integrations are required to achieve the intended outcome.

Building a Business Case

Repository security can affect engineering effort, release schedules, and security operations. A business case should connect those outcomes to the organization's current experience rather than rely on generic cost estimates.

Start by estimating the work associated with a risky component entering development. Consider time spent identifying affected projects, investigating package behavior, removing or replacing dependencies, rebuilding and retesting, documenting the response, and communicating with stakeholders. Include work across security, platform, and application teams.

Then account for the operating effort required to prevent or manage that exposure. This may include policy setup and review, repository administration, exception handling, training, integration, and support. A solution that reduces incident response but creates substantial friction or overhead may not produce the expected value.

A proof of value can help quantify the tradeoff. Compare baseline and pilot measures such as:

- Time security and engineering teams spend reviewing or remediating component issues.
- Number of component requests with a clear allow, block, or quarantine outcome.
- Time to understand and resolve a blocked request.
- Build interruptions and avoidable retesting caused by dependency issues.
- Percentage of component traffic routed through the evaluated control.
- Administrative effort to maintain policies, integrations, and exceptions.

Use these measures to build a realistic estimate of avoided investigation and rework, potential release impact, and ongoing operating cost. Be clear about assumptions, and distinguish measured results from projected benefits.

Make the Trusted Path the Easy Path

When a malicious package reaches a developer or build, teams may have to trace where it went, remove it, and rebuild or retest affected software. Stopping risky components before download can spare developers downstream work.

Sonatype Firewall checks component requests against Sonatype's open source intelligence and your organization's policies. It can block malicious packages, quarantine suspicious requests for review, and give teams context when a request is stopped.

Controlling component decisions at the source gives teams a consistent way to decide what enters development, before packages reach developers or builds. Explore how Sonatype Firewall helps put those decisions into practice at intake.

EXPLORE SONATYPE FIREWALL

sonatype

Sonatype is the company that accelerates agentic software development with confidence. Trusted by thousands of enterprises and millions of developers, Sonatype helps organizations build with confidence by governing the open source, AI-generated, and third-party components that power modern software. As the steward of Maven Central and the company behind Nexus Repository, Sonatype provides unmatched visibility into how software is built, consumed, and secured — helping teams move faster, reduce risk, and ship software with confidence at AI scale. To learn more about Sonatype, please visit www.sonatype.com