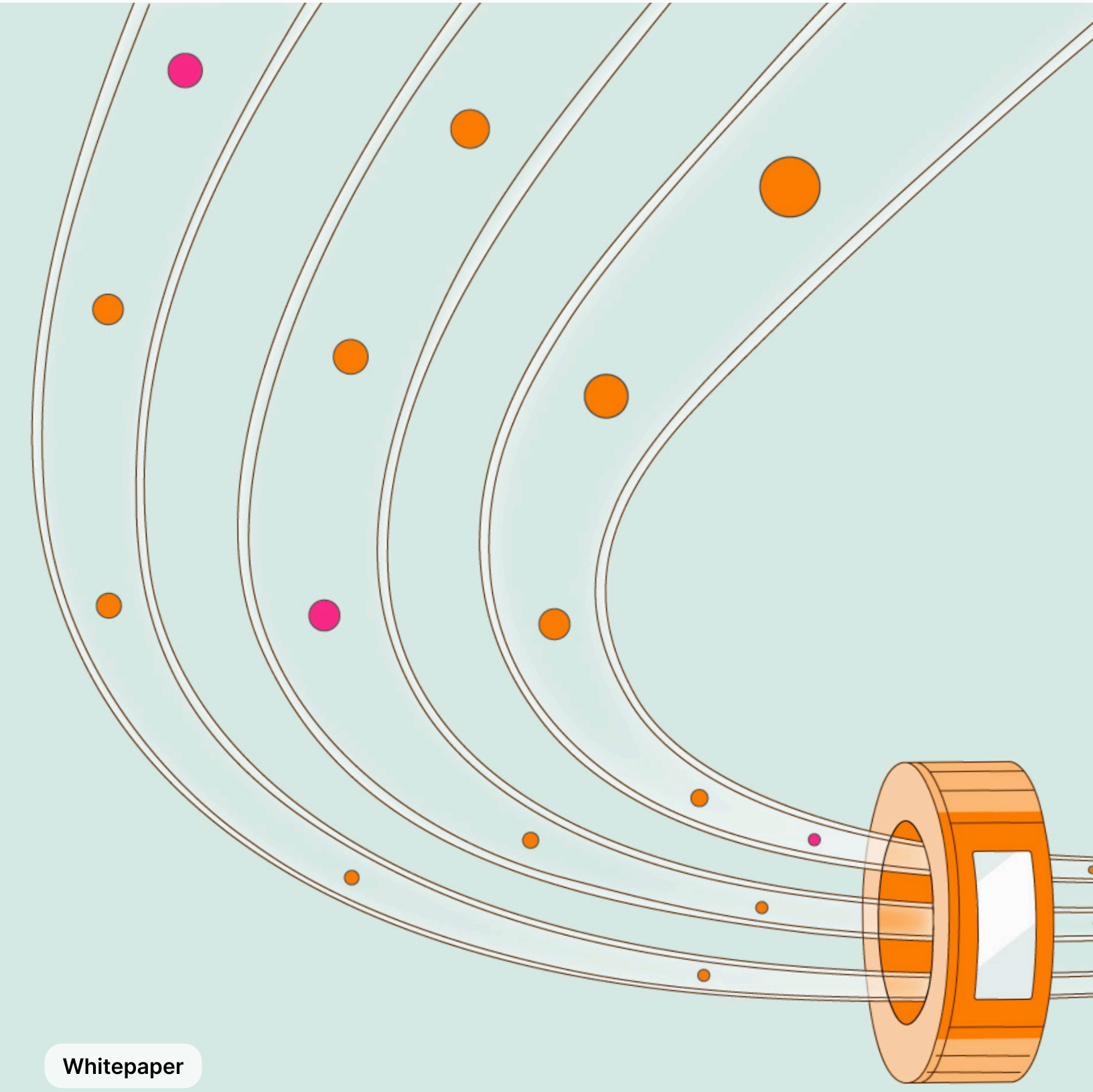
Semgrep

Reachability Analysis

Cutting Through Noise to Identify the Risks That Actually Matter



Whitepaper

Table of Contents

Executive Summary *Page 2*

Introduction *Page 3*

Why Reachability Analysis *Page 3*

Reachability Analysis in the Real World *Page 4*

Usability *Page 4*

Language Support *Page 4*

Onboarding *Page 5*

Precision *Page 6*

Dependency Reachability *Page 6*

Function-Level Reachability *Page 6*

Dataflow Reachability *Page 7*

Reachability Classification *Page 8*

Performance *Page 11*

Conclusion *Page 12*

About Semgrep *Page 12*

Executive Summary

Advancing Supply Chain Security with Code-Aware Reachability Analysis Developers can Trust

The pervasive use of third-party libraries, while essential for development velocity, has introduced the difficult trade-off between addressing potential security risks and distracting engineering teams with unnecessary updates. Traditional Software Composition Analysis (SCA) tools fall short by flooding AppSec and developer teams with an overwhelming volume of vulnerability alerts, leading to wasted effort triaging, alert fatigue, and organizational friction between AppSec and developers.

The evolution in addressing this problem is **code-aware reachability analysis**, that triages vulnerabilities based on the application's code specific usage of the vulnerable function. This shifts the focus from a list of all potential threats to a curated list of actual risks. However, the market offers varying levels of efficacy, and the practical benefits of reachability analysis are highly dependent on its implementation.

Key differentiators for an effective reachability-based SCA solution include:

- **Granularity of Analysis** Basic "dependency-level" analysis only identifies unused packages, while advanced "function-level" analysis screens out unused functions within used packages. Semgrep augments function level analysis with further "dataflow reachability," which analyzes how data flows to a vulnerable function to determine if it can actually be exploited, providing the lowest false positives of any reachability approach.

- **Operational Practicality** Many solutions require disruptive runtime agents or complex per-repository CI/CD configuration, making them difficult to scale. An ideal platform offers seamless, centralized onboarding that does not burden developer workflows or infrastructure.

- **Performance and Coverage** For developer adoption, full-application scans must be fast—under five minutes—to avoid blocking CI/CD pipelines. Furthermore, comprehensive support for multiple programming languages is essential to accurately map dependencies in modern, polyglot applications.

Reachability analysis represents a fundamental leap in software supply chain security. To fully realize its value, organizations must select solutions that move beyond marketing claims to deliver deep, code-aware analysis combined with the performance and ease of use required for enterprise-scale, high-velocity development environments. This enables AppSec teams to focus on genuine risks and developer teams to remediate with confidence, significantly improving security posture without sacrificing velocity.

Introduction

The use of third party libraries has become an inescapable part of the software development process. It helps to accelerate development by providing pre-written, tested code for common functionalities, saving time and cost. But it also complicates the problem of package management. For example, a new version of a library may introduce subtle changes that can cause regressions, and require a backlog of effort to verify that no unintended bugs are introduced, all of which takes significant engineering investment.

Traditional Software Composition Analysis (SCA) tools were supposed to help tackle this problem by allowing application security teams to scan for vulnerabilities in open source dependencies. But in reality, they provided no insight into whether or how an application used those dependencies, leading to a large volume of alerts. So, how do teams make sure that by updating a dependency to try to reduce security risk they're not introducing an engineering risk?

Reachability analysis mitigates this dilemma by offering context around whether the application code is in fact calling the vulnerable dependencies. This enables software teams to deprioritize or even ignore issues, and thus be deliberate about updating their dependencies. It also empowers engineering teams to efficiently prioritize those alerts that yield tangible benefits to the product.

Why reachability analysis?

Traditional SCA tools analyze an application's third-party dependencies for security vulnerabilities. These SCA tools are designed to find *all potential third party vulnerabilities in the application software*. However, most SCA approaches rely solely on version checks for software packages being imported. They don't scan the application source code to see if and how the dependencies inside the package are actually used.

This opacity means that traditional SCA tools mark as vulnerable all packages with vulnerabilities, whether or not they are being used in a vulnerable way *or even being used at all*. This leads to an excessive volume of alerts, as well as a frustrating back-and-forth between security teams that flag SCA findings as risky, and developer teams who then have to explain, for example, that they're not even using those capabilities of the package which the finding marked as vulnerable.

To address this shortcoming, Semgrep was among the first vendors to introduce *reachability analysis* to the market, fundamentally transforming how SCA is used. Reachability analysis determines if the vulnerable part of the open source dependency is used (reachable) by the application, helping security teams and developers understand whether a vulnerability is likely to be exploitable. In sum, *reachability analysis shifts the focus from an overwhelming list of potential threats to a curated list of actual risks, enabling developers to fix what truly matters*.

Since then, leading edge security vendors have added some form of reachability analysis to their SCA offerings. Let's take a look at how different approaches tackle the problem of excess noise and evaluate their efficacy.

Reachability analysis in the real world: *Semgrep vs other vendors*

Reachability analysis might be as close as SCA can get to a silver bullet for false positives - in theory, but in practice several limiting factors can attenuate its effectiveness, with varying implications for security and developer teams. Different solutions can claim to have reachability analysis yet practitioners still see dramatic differences in usability, precision, and performance, among other considerations.

Usability

Language support

One significant hurdle to gathering meaningful results is that reachability analysis is language-dependent. While SCA is based on generalizable heuristics, the implementation requires a deep understanding of each language's semantics, features, and runtime behaviors. The secret is determining whether and how your code interacts with vulnerable third-party dependencies.

Additionally, when an application written in one language uses a dependency in another language, like a Scala application calling Java libraries, a reachability scan that doesn't support a given language would miss the call paths that cross the language boundary, and thus fail to even identify the dependency, let alone determine whether it is reachable. A complex organization with projects in multiple languages would require different parsers and different behaviors for each language supported in order to be able to run reachability-based SCA scans.

Making sure that an SCA solution supports all the languages used by your organization, whether directly or indirectly, is essential for getting reachability insights. Among all the vendors in the market, Semgrep has the most extensive coverage for reachability analysis, with reachability supported in 11 languages, including the most popular ecosystems for the most popular languages.

Language support among reachability-based SCA solutions

Reachability Language Support	Semgrep	Endor	Socket	Snyk
Javascript/Typescript	✓	✓	✓	✓
Java	✓	✓	✓	✓
Python	✓	✓	✓	✓
Go	✓	✓	✓	✗
C#/.NET	✓	✓	✓	✓
Kotlin	✓	✓	✓	✗
Ruby	✓	✗	✓	✗
Swift	✓	✗	✗	✗
Scala	✓	✓	✓	✗
PHP	✓	✗	✓	✗

*This chart is accurate as of October 2025, but is subject to change.

Onboarding

Most SCA tools offering meaningful levels of reachability analysis have one hidden drawback: they either require you to install disruptive runtime agents in your production environment, or perform manual setup *per repository* in order to be able to conduct reachability analysis. Before any application can be screened, the repositories have to be onboarded to the scanning platform, which typically involves configuration, deployment, and subsequent maintenance of scan infrastructure. Once onboarding is finally complete, any new team or repository would require additional onboarding, with further CI/CD configuration, putting additional resource drains on AppSec professionals.

This friction makes SCA tools prohibitively difficult to scale and operationalize across a large organization. Even cloud-native platforms struggle in the real world to seamlessly scan all of the repositories at large enterprises, with customers sometimes waiting years and still not being able to realize the benefits of reachability analysis for which they chose a particular SCA vendor.

With Semgrep's scalable, managed infrastructure, practitioners can onboard repositories automatically and run reachability-based SCA scans, without the need to change existing CI/CD configurations. Instead of adding a Semgrep job or workflow to your CI/CD pipeline, which is hard to scale across large numbers of repos, you could add repositories to Semgrep AppSec Platform *in a few clicks* through its own user interface. SCA scans can then run on the platform's infrastructure instead of your CI/CD infrastructure, saving you CI spend and time spent managing and configuring CI runners. This means even *thousands of repositories can be onboarded in minutes*, and reachability analysis can be performed on every PR, across your entire organization, without the time-consuming struggle of onboarding, configuring, and deploying typical SCA tools.

Precision

Most vendors offering reachability-based SCA solutions primarily use 2 different methods of analysis: rudimentary dependency level analysis, and more advanced function level analysis.

Dependency Reachability

Dependency level reachability analysis, or **dependency reachability**, is the most basic method of determining whether a finding can be deprioritized or ignored. Dependency reachability scans manifest files for listed dependencies and generates a dependency graph that connects them. These are then parsed using import statements to determine which packages are used, and which are not. If the package is never used, it can be marked as *dead*, or not reachable, and therefore the corresponding finding can be ignored.

This approach has crucial limitations, however. First, this kind of analysis requires the use of manifest files, which are metadata files that enumerate every third-party dependency, as well as information about downloadable artifacts. But these are not always comprehensive, in which case AppSec teams are left facing the dilemma that either they demand that their developers generate manifest files, or else forgo the option of reachability analysis itself.

More importantly, even when manifest files are present, dependency reachability looks only as far as the *packages being imported*, not the *functions being used*. This can tell you only which dependencies will never, under any circumstances, be used by your application. That is why dependency reachability results are only useful in identifying which packages *are not reachable*, but they cannot tell you which packages are, in fact, reachable, let alone offer any insight on the exploitability of the functions within those packages.

Dependency reachability identifies which third party packages are imported by an application, and which of those are not used. But it cannot determine whether a package being used has vulnerable functions called by the source code or not.

Some vendors claim to offer reachability analysis, but are limited to answering the rudimentary question "is the dependency used?" In Semgrep's view, this is insufficiently helpful, as engineers still have to invest effort in triaging supposedly "reachable" findings to actually figure out which ones to prioritize.

While there is no canonical definition of reachability analysis, using the term for package level analysis is a generous interpretation. At best, it can only screen out the most obvious findings, without actually revealing actionable context about how your code might reach a given CVE, leaving AppSec teams in the dark about both potential and actual risks within their code.

Function-Level Reachability

To cover the gap left yawning by mere dependency reachability, certain SCA tools also offer **function level reachability** analysis, which looks beyond the dependency graph into your actual application code. In addition to mapping out the dependencies imported by your code, it performs static analysis on your first party code to build a detailed model of which dependencies are actually used within your application. This includes *mapping vulnerabilities back to vulnerable functions*—hence, function level reachability—so that static analysis can target vulnerabilities with higher levels of granularity.

By mapping how each vulnerability may be reached *in the context of your code*, and therefore mark potentially exploitable functions with higher assurance, function-level reachability can reduce significantly more noise than is possible with dependency-level reachability. In some environments, upwards of 90% of findings can be demarcated as unreachable.

Consider the case where a package being imported contains vulnerable functions: function-level reachability identifies the vulnerable functions inside the package, determines if your application is in fact calling the vulnerable functions in question, and therefore flags the package as vulnerable. If any of those criteria are not satisfied, the finding is marked unreachable.

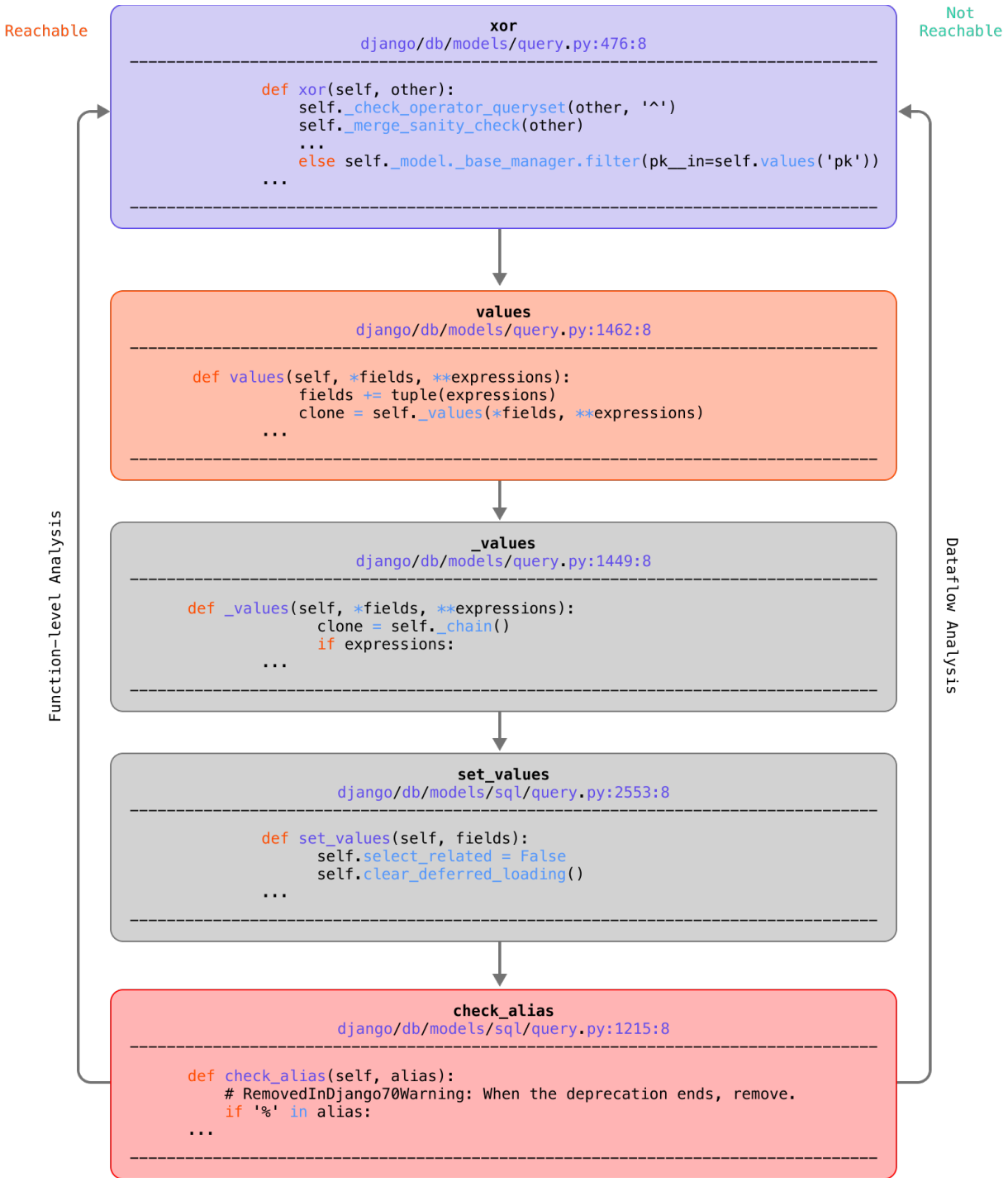
Being able to analyze findings in the context of your code, function-level reachability can tell you not only which dependencies are not reachable (true negatives), but also conclusively tell which are in fact reachable (true positives). But since it cannot identify cases where the function is vulnerable, *but only under certain conditions*, it cannot provide insight on whether a reachable vulnerable function is in fact exploitable. Put differently, a true positive under the reachability lens can be a false positive from the point of view of exploitability.

This is where understanding how data flows between functions helps make that further distinction, and where Semgrep's reachability analysis differs from other approaches on the market.

Dataflow Reachability

There are cases where your code could call a function that contains a vulnerability, but doesn't use it in a vulnerable way. Function level reachability analysis cannot see how the data flows between the function and your code, and would identify this finding as vulnerable, creating a false positive.

For example, consider the following SQL injection vulnerability in Django:



The function `values` calls a vulnerable function `check_alias`. Under function level reachability, `values` will be flagged as reachable, along with any other function that calls `check_alias` (so long as it is publicly exported), for example, `__xor__`.

But dataflow analysis will look further into the function arguments. For example, the `__xor__` function calls `values` (which is, in fact, vulnerable), but it calls `values` with a static string variable. Which means that, in this case, there is no way for an attacker to send data into `__xor__` and have it reach `values`. Thus dataflow analysis will mark the `__xor__` function as unreachable.

This example shows how dataflow reachability can further reduce false positives by understanding function arguments in the context of your code.

Some functions are marked vulnerable, but only under a certain set of conditions. Other SCA tools would mark this finding as requiring upgrade, while dataflow reachability can look into the exact execution path of the application to determine that the vulnerability is, in fact, not reachable.

Built on top of the world's most popular open source SAST engine, Semgrep Supply Chain is the only SCA solution to provide **data flow reachability analysis**, which goes one level deeper than function level reachability analysis. Crucially, because of the way it is architected, Semgrep provides this level of insight without the associated costs and drawbacks of traditional SCA scanners. While other tools have to build out the entire dependency graph to explore application paths to find exploitable code, Semgrep leverages its SAST engine to perform deep static analysis on the application code to *map the exit points of the source code to the entry points of a CVE*.

This allows Semgrep to perform not only function level reachability analysis, but go one level deeper, onto the dataflow level. It can not only reveal *if* your application is calling a vulnerable function, but also *how exactly* the function is being used.

We'll look more into this level of insight in the following section.

Reachability Classification

Regardless of how reachability analysis is performed, the main purpose of SCA boils down to this: *which packages do I need to upgrade* to minimize risk to my application? All flavors of reachability analysis aim to provide AppSec engineers insight and guidance to help make that determination.

One of the unique benefits of having the Semgrep Pro Engine as a foundational core is the ability to classify reachable findings into further subcategories that have different implications for AppSec. Semgrep analysis engine categorizes findings into the following groups:

Reachable: A finding is reachable if there's a vulnerable function call or vulnerable package in use. The finding should be addressed as soon as possible.

Malicious Dependency

A finding that indicates the use of a dangerous package, or dangerous version of a package, that is designed to compromise systems. These packages are inherently dangerous, and thus need to be immediately updated.

Reachable in Code

A finding is reachable in code if there's a code pattern in the codebase that matches the vulnerability definition. This is where dataflow analysis confirms that a vulnerable package is being called in a vulnerable way by your code.

Always Reachable

A finding is always reachable if it's something Semgrep recommends fixing, regardless of what's in the code.

Needs review: A finding that requires manual triage and review; follow the instruction provided.

Conditionally Reachable

A finding is conditionally reachable if Semgrep finds a way to reach it when certain external conditions are met. For findings that have no reachability information, this classification helps simplify the task of guidance by pinpointing the precise combination of conditions that can render a dependency actually vulnerable. Conversely, the absence of those conditions means the dependency, although nominally vulnerable, does not pose additional risk to your application.

No Reachability Analysis

A finding that cannot be scanned for reachability because of factors extraneous to static code analysis. For example, the function is only vulnerable if there is no external firewall configured on your application, or the vulnerability only works in a Unix OS. In these cases, static analysis on its own cannot determine exploitability with certitude, so runtime or pentesting would be needed to ascertain how the application code traces to a CVE.

The most basic form of analysis will suggest you upgrade packages for all findings that superficially appear to be potentially risky. More sophisticated analysis will comb through the findings with increasing depth to determine with confidence which ones can be safely ignored, and which ones warrant remediation. The more advanced the analysis, the smaller the set of findings requiring fixes and the greater the degree of confidence. Semgrep is unique among SCA solutions to offer this level of granularity in its findings, providing software teams rich context to inform their prioritization decisions.

Performance

While reachability analysis does reduce the noise in vulnerability findings, you often pay a cost in increased scan times to get that additional data. So how do you make sure developers aren't waiting on slow scans? Customers shopping for SCA solutions have made it clear that scan speed determines developer adoption: if scans are slow, PRs can be delayed or even blocked, and subsequently impact developer velocity. Whereas a scan that takes a few minutes to complete can be easily incorporated into the developer workflow, often completing before other CI jobs.

Many solutions that purport to offer function-level reachability analysis do so *only in full scans*, which in most cases take long enough to perform that they disrupt developer workflows and impinge on CI/CD pipelines. So in order to offer an SCA option that is fast enough to be used by devs, these solutions recommend the use of *diff-aware scans*, which accelerate scan times by only scanning the files modified by a commit in a feature branch. However, for most SCA vendors, these partial scans do not provide function level reachability analysis, sacrificing depth of analysis in pursuit of speed. What these vendors provide instead is package level analysis, with its limitations, as explained above.

For example, some vendors caveat their noise reduction capabilities by disclaiming, e.g.

Socket

While full application [function level] analysis provides better noise reduction [than their lower tier reachability offerings] and richer context for triaging reachable vulnerabilities, it comes with some trade-offs: ***it requires manual setup and is generally slower, as the static analysis is compute-intensive.***

ENDOR LABS

Full [function level] scans perform dependency resolution, reachability analysis, and generate call graphs for supported languages and ecosystems. This scan enables users to get complete visibility and identifies all issues dependencies, call graph generation before merging into the main branch. ***Full scans take longer to complete, delaying PR merges***

It is important to note that the noise reduction figures marketed by other vendors are based on full application scans, which being slow and compute intensive, are not likely to be used in typical development workflows. This is why most vendors provide the less accurate dependency reachability analysis options as "fast scan" offerings to be incorporated into dev workflows, *trading precision for performance and usability*. Thus, the SCA scans actually likely to be used by devs on a regular basis will have all the noise and false positives of rudimentary, manifest-based dependency reachability. Which translates to 300% more false positives* in the real world than customers might expect based on the marketing of full scans as the de facto option.

Semgrep Supply Chain was built to optimize for developer experience first, using parallel processing and diff-aware caching that avoids redundant code parsing, to achieve a median scan time for *full scans in under 5 minutes*. Semgrep Supply Chain is, in fact, the only solution that provides full application reachability analysis, with dependency resolution, reachability classification, at speeds that devs can use in high velocity workflows.

*https://docs.socket.dev/docs/reachability-analysis#reachability-maturity-levels

Conclusion

Reachability analysis has helped SCA tools evolve past their rudimentary iterations which would merely list all dependencies, drowning AppSec in alerts, to their present state where significant noise reduction is possible due to the ability to contextualize findings within your codebase. However, different approaches to reachability produce variable results, while the actual benefits of reachability are tricky to realize in today's high-velocity dev environments. It's important to consider ease of onboarding, scan time performance, language support, as well as granularity of results to make sure that the potential benefits of reachability analysis can in fact be actualized. With the right code-aware SCA solution in place, development teams can write more secure code, and AppSec teams are able to focus on the real risks that can otherwise get drowned out in the noise of low-resolution scanning tools.

About Semgrep

Semgrep is the leading code security platform for builders – helping teams catch, flag, and fix real issues before they ship, with security that learns as you build. Its developer-first platform unifies SAST, SCA, and secrets detection, embedding security directly into the development workflow so protection begins where code happens. Semgrep combines deterministic static analysis with AI reasoning that powers detection, triage, and remediation to help teams uncover real vulnerabilities, prioritize reachable risks, and fix issues faster.

Backed by Menlo, Felicis, Lightspeed, Redpoint, and Sequoia Capital, Semgrep is trusted by global organizations, including Snowflake, Dropbox, and Figma.