The Qwiet AI Platform

Agentic AI, Code Property Graph,
and AutoFix for Secure Code Development

Table of Contents

Executive Summary

Modern software development faces a paradox: rapid release cycles demand speed, yet security traditionally slows development. Qwiet AI's preZero platform addresses this challenge by blending **Agentic AI** (a multi-agent artificial intelligence approach), the **Code Property Graph (CPG)**, and **AutoFix** capabilities to deliver secure code without sacrificing agility. In essence, preZero acts as an autonomous AppSec team that **finds, prioritizes, and fixes vulnerabilities in code** in a fraction of the time required for manual processes. Key innovations include a multi-agent AI workflow that collaborates like specialized security engineers, a unified code analysis graph that provides deep contextual insight, and AI-driven remediation that generates validated fixes aligned with industry best practices (OWASP, NIST, ISO 27001).

This whitepaper provides a technical deep dive into how these components work together to improve the software security lifecycle, reduce false positives, accelerate mean-time-to-remediation (MTTR), and support comprehensive security practices (SAST, SCA, IaC scanning, SBOM, container security). It also highlights real-world results from an enterprise proof-of-concept, demonstrating significant ROI and efficiency gains. What's powerful is that a single scan process handles all of these.

In summary, the Next Generation, AI-Native Qwiet AI solution enables organizations to ship secure code faster by automatically detecting critical vulnerabilities, understanding their actual risk (reachability and exploitability), and autonomously and predictively providing tested fixes, all integrated into development workflows. Security leaders, AppSec engineers, and software developers/engineers gain a scalable AI-powered assistant that augments their capabilities, reduces alert fatigue, and ensures compliance with key security standards. The following sections detail the implementation and benefits of this approach, with technical rigor and practical insights.

AI Code Generation: The Need for Smarter, Faster AppSec

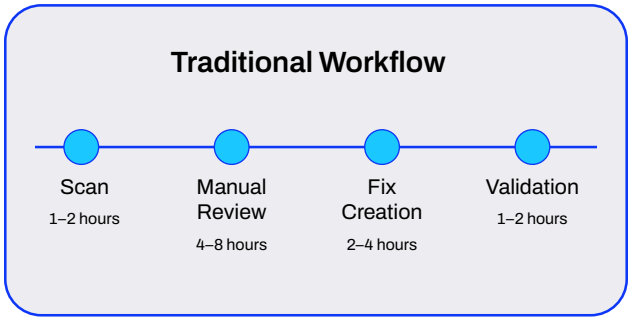
Enterprises today grapple with an overwhelming volume of security findings and a shortage of AppSec experts to triage and remediate them. And now, with the introduction of AI coding, the code tsunami is on the precipice of disaster without objective cyber oversight. Combine this fact with traditional static analysis tools producing a deluge of alerts (often with 50-90% false positives) that strain development teams, and we are setting ourselves up for massive and systemic failure.

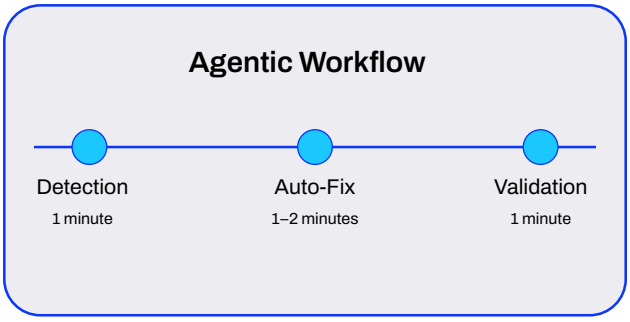
Manual review and fixing can stretch over days, forcing teams into a lose-lose choice: delay releases or push code with known issues. Security and development often seem at odds in a **"speed vs. security" dilemma.**

The Code Property Graph and Agentic AI (a team of AI "personas") are introduced into the software security lifecycle by Qwiet AI's solution to address this issue. This innovative platform provides:

- **Context-sensitive Prevention:** AI agents continuously scan code, configurations, and dependencies for vulnerabilities before they reach production. Each finding is enriched with context from the entire codebase to assess actual risk.
- **Fixing at the Speed of AI:** Instead of waiting on humans, the platform's agents propose fixes in real-time, cutting remediation from hours or days to minutes. In one comparison, an end-to-end patch cycle dropped from ~8–16 hours to ~2–4 minutes using Qwiet's autonomous workflow.
- **Atomize Noise and False Positives:** reduces alert fatigue and allows developers to concentrate on actual threats by utilizing deep code analysis and reachability logic to eliminate theoretical issues that are not exploitable.
- **Best Practices Compliance:** Generated fixes and recommendations follow security best practices (OWASP Top 10, NIST Secure Coding Guidelines) and are audit-ready for compliance frameworks like ISO 27001.
- **Enterprise Integration:** The solution plugs into existing development pipelines (IDEs, CI/CD) to provide immediate feedback and even block risky code from being merged while operating in the background, so as not to impede developers.

In the following sections, we explore the core technological pillars of Qwiet AI – the agentic AI architecture, the Code Property Graph, and the AutoFix system – and how they deliver a step-change in application security efficacy and efficiency.





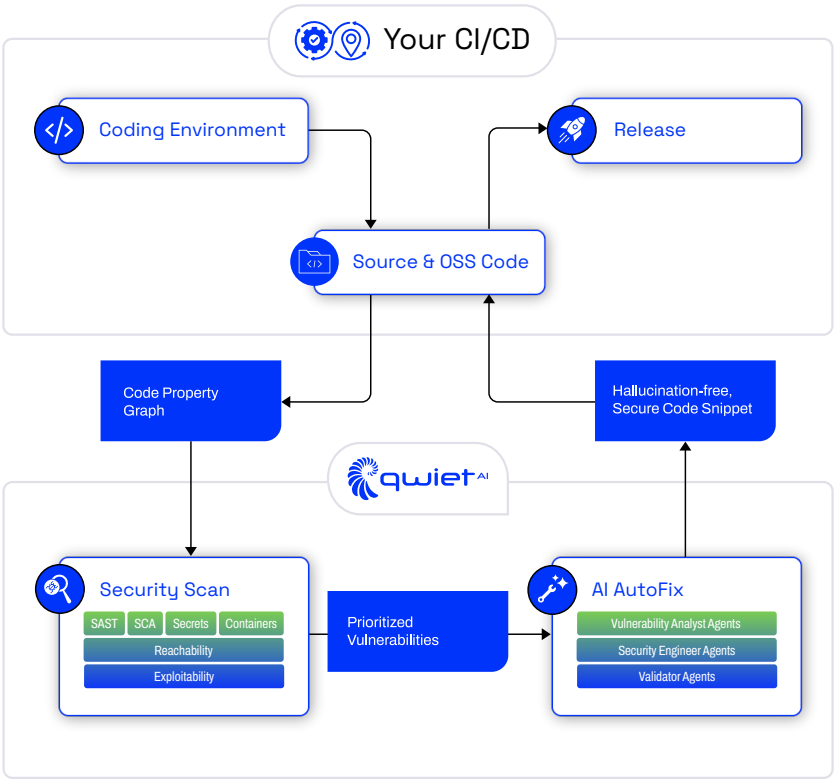
Time Reduction: 8–16 hours → 2–4 minutes

Manual Effort: 6–12 hours → 0 minutes

Agentic AI: Multi-Agent Collaboration in the Security Lifecycle

Qwiet AI's **Agentic AI** is an approach where multiple specialized AI agents work in concert, emulating a team of security experts, to secure the software development lifecycle. Each agent has a defined role (or persona) with specific responsibilities, and they collaborate to identify and remediate vulnerabilities with minimal human intervention. This distributed intelligence model brings several advantages in coverage, speed, and accuracy:

- **Parallel & Specialized Analysis:** In an agentic AppSec environment, each AI agent focuses on a distinct security aspect (e.g., one on threat modeling, another on testing, another on fixing). By operating in parallel or sequence as needed, vulnerabilities are found faster and understood from multiple angles, ultimately converging on a fix.
- **Contextual Decision Making:** Agents don't work in isolation - they share insights. For example, one agent referencing historical fixes or known exploit patterns can inform another agent writing a patch. Because each agent considers the local code snippet and global context (dependency versions, known CVEs, past commits), the system's decisions are context-aware and less prone to error.
- **Reduced Alert Fatigue:** With AI sifting through the noise first, developers and security engineers are presented only with meaningful findings and concrete solutions. The multi-agent system weeds out false positives and low-risk issues through consensus and cross-verification, so teams aren't stuck in endless triage.
- **Research-Backed Effectiveness:** The multi-agent approach is backed by academic research and real-world testing, which shows that iterative, cooperative AI analysis yields higher fix accuracy. Each agent's output is validated by others (e.g., a fix is checked by a testing agent), which improves correctness and reliability over a single-pass analysis. This **self-verification and cross-verification** mechanism catches mistakes (like an incomplete fix or a "hallucinated" code change) before they reach the developer.



Specialized AI Personas and Workflow: Qwiet AI's platform employs a cast of AI personas that mirror the AppSec roles in a security team. Below are key agents in the agentic workflow and how they collaborate:

- **Threat Analyst (AI):** This agent acts like an offensive security expert. Its primary task is to scan the code (and IaC, dependencies, etc.) for potential weaknesses and craft ***real-world exploit payloads*** that an attacker might use. Approaching findings from an attacker's perspective ensures that the flagged vulnerability is not just a theoretical problem but a clear exploit scenario.

  Value add: The Threat Analyst agent ensures the process begins with an attacker mindset, so fixes later will thwart real exploitation attempts, not just quiet static analysis warnings.

- **Test Engineer (AI):** Once a potential vulnerability and exploit scenario is identified, the agent generates automated tests (or proof-of-concept exploits) to validate the issue. It translates the payload from the Threat Analyst into structured test cases (e.g., unit tests or integration tests) that will fail against the current code, demonstrating the presence of the vulnerability.

  Value add: Before any fix is applied, we have a concrete failing test that confirms the vulnerability. This agent-driven test will later verify if a proposed fix resolves the problem, ensuring no "half-measures or untested patches" slip through.

- **Refactor/Remediation Engineer (AI):** Acting like a software engineer focused on bug fixes, this agent proposes an initial code change to remediate the vulnerability. It uses the context from the Code Property Graph and guidelines from known best practices (for example, if the issue is an SQL injection, it might suggest using parameterized queries per OWASP recommendations). The fix is crafted to address the failing test and fit the project's coding patterns and style.

  Value add: The Remediation agent dramatically speeds up fix creation. Instead of a developer spending hours researching and writing a patch, the AI produces one in minutes, often informed by *OWASP and NIST secure coding standards* to ensure the solution is compliant and high-quality. This agent effectively encodes remediation expertise (like knowledge of common vulnerability fixes) into the development workflow.

- **Validation & Evaluation Agents (AI):** After a proposed fix, additional agents review and verify it. The **Test Agent** re-runs the earlier tests (and possibly generates new edge-case tests) to check if the vulnerability is fixed. A dedicated **Evaluator** agent acts as a code reviewer, scrutinizing the fix for any gaps or side effects. It might identify missing scenarios or suggest improvements. If the fix is inadequate, the issue is fed back to the Remediation agent for another iteration this loop continues until the vulnerability is conclusively resolved and all tests pass.

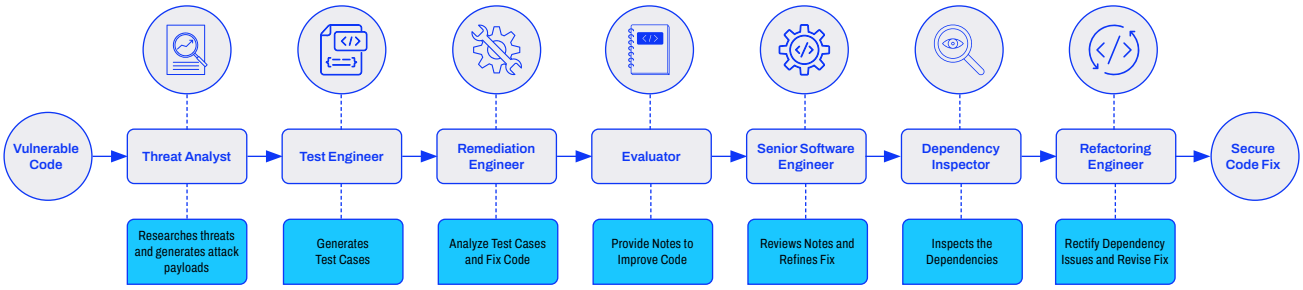
  Value add: These checks ensure the fix is not just a superficial patch. The Evaluator encourages iterative refinement, catching cases where a fix might introduce a new bug or not fully solve the problem. This results in a solution that a human team lead (if involved) can trust.

- **Senior Engineer & Dependency Auditor (AI):** Finally, a "senior" AI agent can integrate all feedback to polish the fix, addressing edge cases, performance, or style issues as needed. Additionally, a **Dependency Auditor** agent (nicknamed "Hal" in the system) reviews any external libraries or packages involved in the fix. Hal is essential for evaluating the security and reliability of a dependency when an AI suggests adding or updating it. For example, Hal can check if it is a known safe library and not a malicious or outdated package. It uses a scoring system (considering known vulnerabilities, version pinning, popularity, maintenance activity, etc.) to flag suspicious dependency changes. If something is amiss (perhaps the AI suggested an unmaintained package), the system will adjust the fix or notify developers accordingly.

Value add: This prevents the AI from inadvertently introducing new risks via third-party components (a form of hallucination mitigation) and ensures the final code meets the project's security and quality standards at all levels.

Through this multi-agent collaboration, Qwiet's Agentic AI essentially replicates a complete secure development workflow autonomously: from finding a bug, proving the bug, fixing the bug, to double-checking the fix. All of these steps can occur within minutes or even seconds. The outcome is a drastically improved software security lifecycle: vulnerabilities are detected and resolved continuously and rapidly, often without human intervention, but with a level of rigor and context that equals or surpasses manual efforts. The platform delivers DevSecOps on autopilot, a continuous loop of detection, remediation, and validation that keeps pace with modern CI/CD.

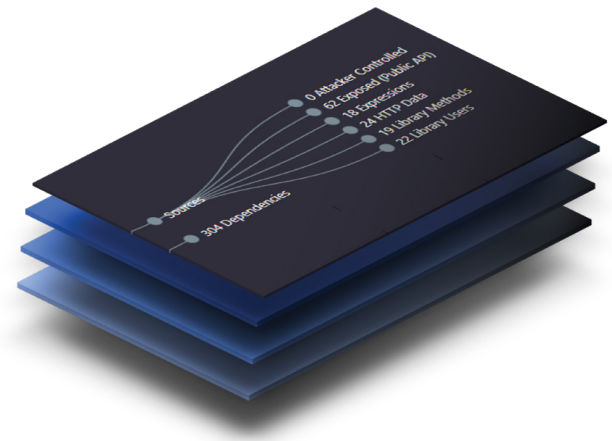
Guided Agentic Workflow



Code Property Graph (CPG): Unified Contextual Vulnerability Detection

At the core of Qwiet AI's analysis engine is the **Code Property Graph (CPG)**, an advanced code representation technology. A CPG is a graph-based intermediate representation of software that **unifies syntax, control flow, and data flow information** into a single structure. This unified model provides a comprehensive view of the code, enabling more powerful and context-aware security analysis than traditional static analysis tools.

What is a Code Property Graph? In technical terms, a CPG merges an Abstract Syntax Tree (AST), a Control Flow Graph (CFG), and a Program Dependence Graph (PDG) into one property graph. Nodes in this graph represent program elements (e.g., functions, variables, library calls), and edges represent relationships (e.g., "A calls B", "X flows into Y"). For example, a CPG will capture not only the call to `exec()` in the code (AST structure) but also how user input might reach that call (data flow) and under what conditions (control flow). With this rich web of relationships, security queries and AI agents can traverse the graph to understand an issue in context: where did the tainted data come from? What path does it take? Is there an authentication check before a sensitive operation? All of that can be inferred from the CPG.



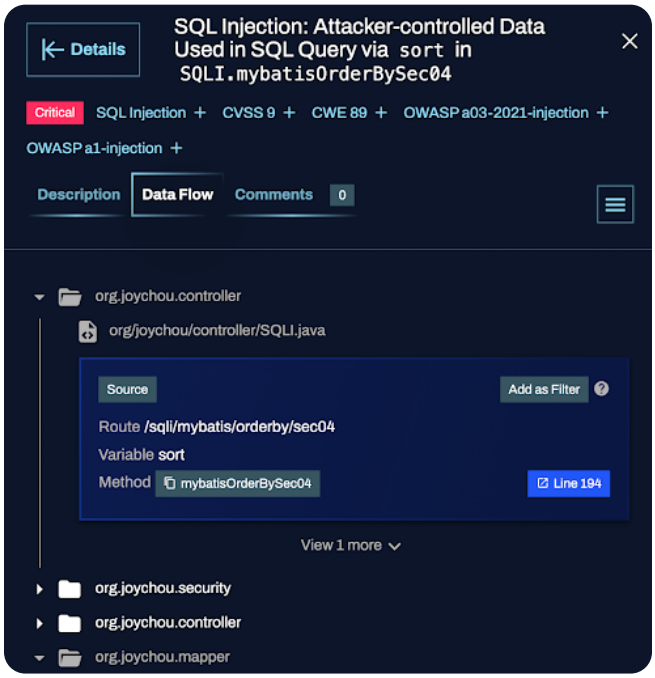
Key benefits of CPG in Qwiet AI:

- **Unified Representation for Deep Analysis:** Because the CPG contains the full spectrum of code relationships, Qwiet's analysis isn't siloed to pattern matching or shallow AST checks. It can perform *contextual vulnerability detection* that considers the interplay of components. The unified graph allows detection of complex, multi-step exploits that might involve multiple functions or files, things like multi-hop injection flaws or logic bugs that span across modules. These are often missed by traditional SAST scanners that lack holistic visibility. In contrast, Qwiet's CPG approach can, for instance, identify that *user data from an API endpoint flows through several transformations and eventually reaches a database query without proper sanitization,* pinpointing a SQL injection that a line-by-line scanner might miss.

- **Improved Accuracy, Fewer False Positives:** Traditional static analysis tends to over-flag issues because it often cannot discern which findings are exploitable. The CPG improves accuracy by incorporating data flow reachability and conditions. Quiet's engine can check if a suspected vulnerability is reachable by any execution path. If not, that finding can be deprioritized or suppressed as a false positive. This deep code context mapping **significantly reduces false positives** compared to legacy tools. In practice, organizations have seen up to a 50% reduction in false-positive alerts by using reachability context. In some cases, even 90% of legacy findings could be filtered out as noise. This precision means developers spend less time chasing ghosts and more time fixing real problems.

- **Scalability and Speed:** The CPG approach is highly optimized despite its depth. Qwiet AI's implementation of CPG analysis delivers results **much faster** than conventional static scanners – **up to 5× faster** in many cases. One reason is that once the code is parsed into the graph, many security queries (or AI agent analyses) can be executed efficiently in parallel on the graph database. The platform is built for modern codebases, efficiently handling large repositories and mono-repos. In a head-to-head proof-of-concept, Qwiet's platform performed comprehensive scans in ~90 seconds, whereas a competing SAST tool took significantly longer for the equivalent analysis. This speed enables security to run *per commit or on every build* without bogging down CI pipelines – a key enabler for DevSecOps.

- **Querying and Extensibility:** The property graph model allows complex security rules or queries to be written in a high-level way (using a domain-specific query language). Security researchers can formulate custom rules (for example, to find a company-specific insecure coding pattern) by traversing the graph. Qwiet has open-sourced aspects of their CPG and its specification, indicating an extensible design. The analysis engine allows for continuously expanding the knowledge base of known vulnerabilities. Additionally, it supports the integration of enterprise-specific checks into the existing graph.



Qwiet AI provides a ***single source of truth*** for its detection agents and remediation logic by unifying code structure and behavior into the CPG. **This comprehensive understanding of the code empowers the Agentic AI to make wise decisions.** For instance, the Remediation agent uses the CPG to ensure a fix covers all execution paths, and the Threat agent uses it to locate all injection points. The CPG essentially acts as the brain's memory for the AI agents, giving them a holistic picture of the application they are securing.

Moreover, the CPG underpins code vulnerability scanning (SAST) and other analyses like dependency analysis and IaC scanning. It can incorporate nodes for dependency versions or cloud infrastructure resources and connect them to application code where relevant. This unified knowledge graph is a foundation that makes Qwiet's multifaceted scanning (explained in a later section) possible under one platform.

AutoFix: AI-Driven, Context-Aware Code Remediation

One of Qwiet AI's most groundbreaking capabilities is **AutoFix**, which automatically generates and applies fixes for detected vulnerabilities using AI. AutoFix is powered by Large Language Models (LLMs) that have been trained on vast amounts of code and security patches, combined with the contextual knowledge from the Code Property Graph. What sets Qwiet's AutoFix apart is its ***agentic workflow*** and emphasis on validated, safe fixes.

How AutoFix Works: When the scanning phase (aided by the CPG and Agentic AI analysis) identifies a vulnerability, the AutoFix system kicks in as described in the agent workflow earlier. The Remediation AI agent (Refactor Engineer persona) crafts a candidate patch for the issue, and then other agents/test cases validate that patch. This process might iterate a few times to converge on a fix that ***eliminates the vulnerability and doesn't break the app's functionality.*** Throughout this, the LLMs are orchestrated to perform specific tasks. One might be prompted to generate a secure code snippet given the context, another to double-check that snippet against a coding standard, etc.

Key features and benefits of Qwiet AI AutoFix:

Context-Aware Fixes via CPG: The AI leverages the Code Property Graph (CPG) to comprehensively understand the vulnerability's context and surrounding code. This includes analyzing call stacks, data flows, and project-specific patterns to suggest a fix. For example, if the vulnerability is a misconfigured cloud resource in Terraform (IaC), the AI considers the broader cloud context; if it's an insecure use of a library function, it checks how that function is used elsewhere in the project. The result is a tailor-made fix for the specific codebase and vulnerability instance, rather than a one-size-fits-all patch. Such context-aware remediation helps ensure the fix integrates smoothly into the existing code and logic.

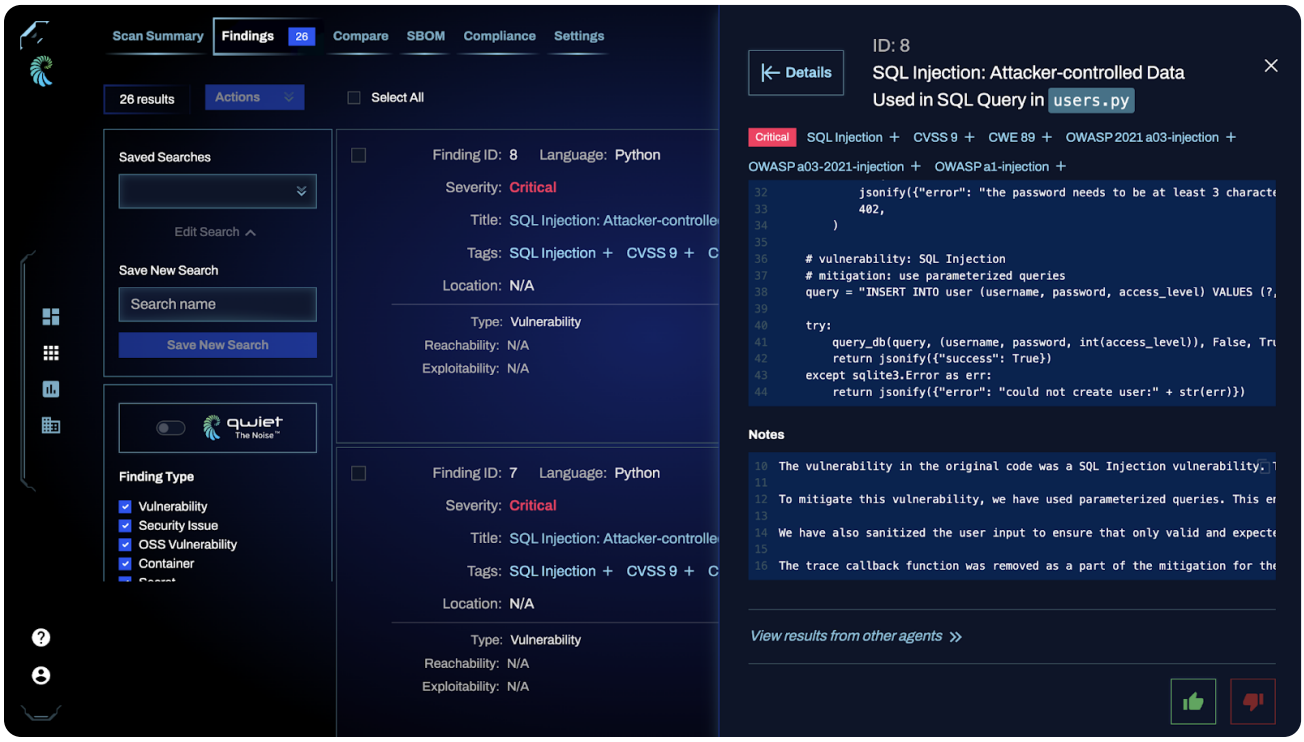
Few-Shot Prompting with Real Examples: The LLMs in AutoFix are guided with ***few-shot prompting***, meaning they are given examples of real-world vulnerabilities and their correct fixes to steer them toward an effective solution. By seeding the AI with known patterns (e.g., how a similar SQL injection was fixed in another context), Qwiet's system increases the precision of generated patches. This technique reduces the chance of producing irrelevant or insecure code changes.

Multi-Agent Verification = Higher Confidence: As described, after an initial fix is proposed, other AI agents test and critique it. This **iterative loop** continues until the fix passes all checks. Essentially, AutoFix doesn't unquestioningly trust a single AI's output – it uses an ensemble of AIs to reach consensus. The multi-agent remediation approach eliminates redundant and incorrect fixes by design, which boosts confidence in the final remediation. For instance, if one agent "thought" the issue was fixed but another agent's tests show a remaining flaw, the fix will be adjusted. This approach yields more patches than a single-pass auto-fix would produce.

Auditability and Compliance Built-In: The AI agents produce a chain-of-thought log that records every step of their process. This allows for **audit-ready fixes** that give human reviewers and auditors confidence that the vulnerability has been appropriately addressed. The final fix explains why the change was made and offers evidence (passing tests) that it functions correctly. Moreover, AutoFix's suggestions are aligned with industry standards, ensuring, for example, that a fix for an injection flaw is consistent with OWASP Top 10 mitigation techniques or that cryptography changes comply with NIST guidelines. The platform explicitly highlights that it ensures compliance with OWASP, NIST, and ISO 27001 in its remediation output. AI can assist organizations in maintaining coding standards and generating documentation to demonstrate compliance with frameworks such as NIST SP 800-218 and ISO 27001 controls. This is especially valuable for organizations facing regulatory pressures.

Asynchronous, Cloud-Based Processing: The analysis and code generation done by AutoFix is handled asynchronously in the background within Qwiet AI's secure cloud (a virtual private cloud environment). This offloads the heavy lifting from the developer's local machine, eliminating wait times and allowing them to continue working while patches are generated and validated. Developers can continue working, and when the AutoFix is ready, it delivers the proposed patch (e.g., via a pull request or as a suggestion in the UI). Qwiet's cloud also means sensitive code does not leave the secure environment – data governance policies (encryption, access control) are enforced during AI processing, addressing concerns about code privacy. The asynchronous model ensures that even though multiple agents and possibly large models work, it doesn't bottleneck the developer's workflow; results come when ready, typically within minutes.

Minimal Manual Effort, "Fixes as a Service": AutoFix can diagnose, fix, and test vulnerabilities in minutes, reducing the manual remediation workload for many issues to almost nothing. This AI-driven process can save a team 6 to 12 hours of combined effort, and depending on company policy, can even automatically apply the ready-to-merge solution. The heavy lifting of creating a correct fix is done, dramatically lowering the mean-time-to-remediate and freeing developers to focus on new features or complex architectural tasks rather than patching bugs. Developers maintain final control and can review, test, and tweak the fix.



To visualize the AutoFix process, consider a simple but common scenario: an API endpoint in a web application has a SQL injection vulnerability due to string concatenation in a SQL query:

- The **Threat Analyst agent** identifies the unsafe query and generates an example input (payload) that would exploit it (e.g., a ' OR '1'='1 SQL injection string).
- The **Test Engineer agent** creates an automated test case that sends this malicious input to the API and expects to observe some injection effect (like a modified query result or error), confirming that the vulnerability is real.
- The **Remediation agent**, using the CPG, sees that the vulnerable code is using raw concatenation to build the SQL query. It cross-references OWASP best practices and suggests using parameterized queries or prepared statements from the application's database library. It generates a code diff that replaces the string concatenation with a safe API call, escaping or binding the input correctly.
- The **Test agent** runs the previously failing test against the modified code. The malicious input no longer causes unintended behavior – the test will pass if the vulnerability is fixed. It may also run regression tests (if available) to ensure nothing else broke.
- The **Evaluator agent** checks the diff to ensure the fix covers all similar instances (if the function is used elsewhere, are those safe?) and that no new flaw was introduced. Suppose the fix introduced a new library called the Dependency auditor (Hal), which would verify that this library is allowed and secure.
- Once all checks are green, the platform can automatically commit the fix or open a pull request with the changes, including a summary: e.g., "AutoFix applied to prevent SQL injection in UserDAO.java using prepared statements (compliant with OWASP recommendations)." The developer can then quickly scan this, see the tests passed, and confidently merge it.

This example demonstrates the **iterative and validated nature** of AutoFix. It's not simply search-and-replace or a single prompt to an AI; it's a governed workflow that yields a vetted solution. By using multiple LLMs and agent personas, Qwiet AI's AutoFix achieves what single-agent or naive auto-fix solutions struggle with: ***trustworthiness***. Enterprises can't afford automated fixes that break things or miss subtle issues, which is why the agentic approach (with chain-of-thought reasoning and cross-verification) is so powerful. It's worth noting that this approach has been shown to boost success rates in automated bug fixing significantly (as cited by research, e.g., Meng et al. 2024 for multi-step reasoning).

In summary, AutoFix turns the output of static analysis into actionable, validated remediation, effectively closing the loop from detection to resolution. This transforms AppSec's role from finding problems to fixing them at scale. It enables a shift where security teams spend more time defining policies and reviewing high-level issues, and routine bugs are handled automatically in the cloud.

Real-World Impact: Case Study Insights and ROI

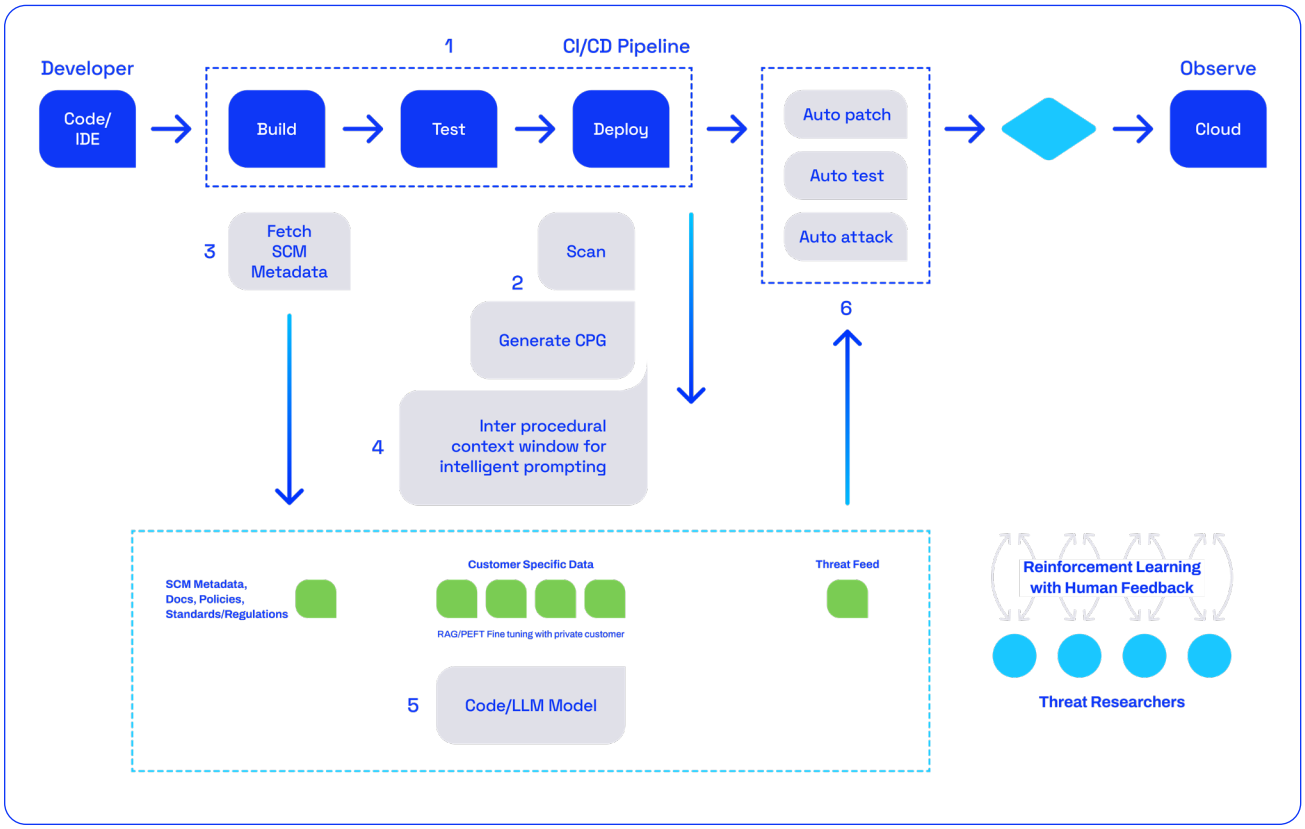
To evaluate the effectiveness of Qwiet AI's solution, consider the results from a recent **proof-of-concept (PoC)** with a large global enterprise (a multi-national telecommunications provider). This PoC, conducted on the enterprise's application codebase and pipeline, demonstrated clear efficiency and risk reduction gains. All company-specific data has been anonymized, but the patterns of improvement are broadly applicable:

- **Unprecedented Noise Reduction:** The PoC showed that using Qwiet's prioritization and reachability engine, the enterprise could filter out the vast majority of findings from their legacy scanning tools as

noise. Approximately **90% of the conclusions produced by the traditional tools were deemed *not exploitable in the context of the application.*** These false positives or minor issues would consume developer time without a security payoff. Since each finding took roughly ~2 hours of a developer's time to investigate and remediate on average, this noise reduction translates to a huge productivity saving. Qwiet AI returned that time to the development team by focusing only on the ~10% of issues that mattered.

- **Reachability & Risk-Based Prioritization:** The concept of *reachability* proved pivotal. Qwiet AI's CPG-based reachability analysis could distinguish between merely theoretical vulnerabilities (e.g., dead code or uncalled vulnerable library functions) and those that were truly reachable at runtime and posed a risk. By categorizing and filtering findings based on execution path reachability, the platform enabled the security team to prioritize **exploitable vulnerabilities first**. The PoC results indicated developers could focus remediation efforts on the critical 10% and address them faster, rather than wading through a backlog of 100% of issues. This approach **reduces remediation cost and time** by avoiding work on non-issues and accelerating fixes for real threats. One slide from the PoC quantified that focusing on reachable vulnerabilities can minimize time spent on false positives by over 90%, optimizing resource allocation and saving significant costs.

- **Improved Scan Efficiency and Coverage:** The enterprise observed that Qwiet's scanning produced more relevant results and did so faster and more frequently. Scans that previously might run in hours or had to be limited to once a day were replaced with near real-time scanning (~90 seconds per scan in this case). This seamless integration into their CI/CD pipeline provided nearly instant security feedback after code check-ins. The CPG's deeper analysis allowed Qwiet to identify specific complex vulnerability patterns (like multi-step data leaks) that the legacy tool missed, without sacrificing coverage. The combination of **faster scans and more brilliant results** led to a higher fix rate; developers were more inclined to address issues when they arrived promptly and with evident importance, rather than a deluge of hundreds of low-quality alerts later.

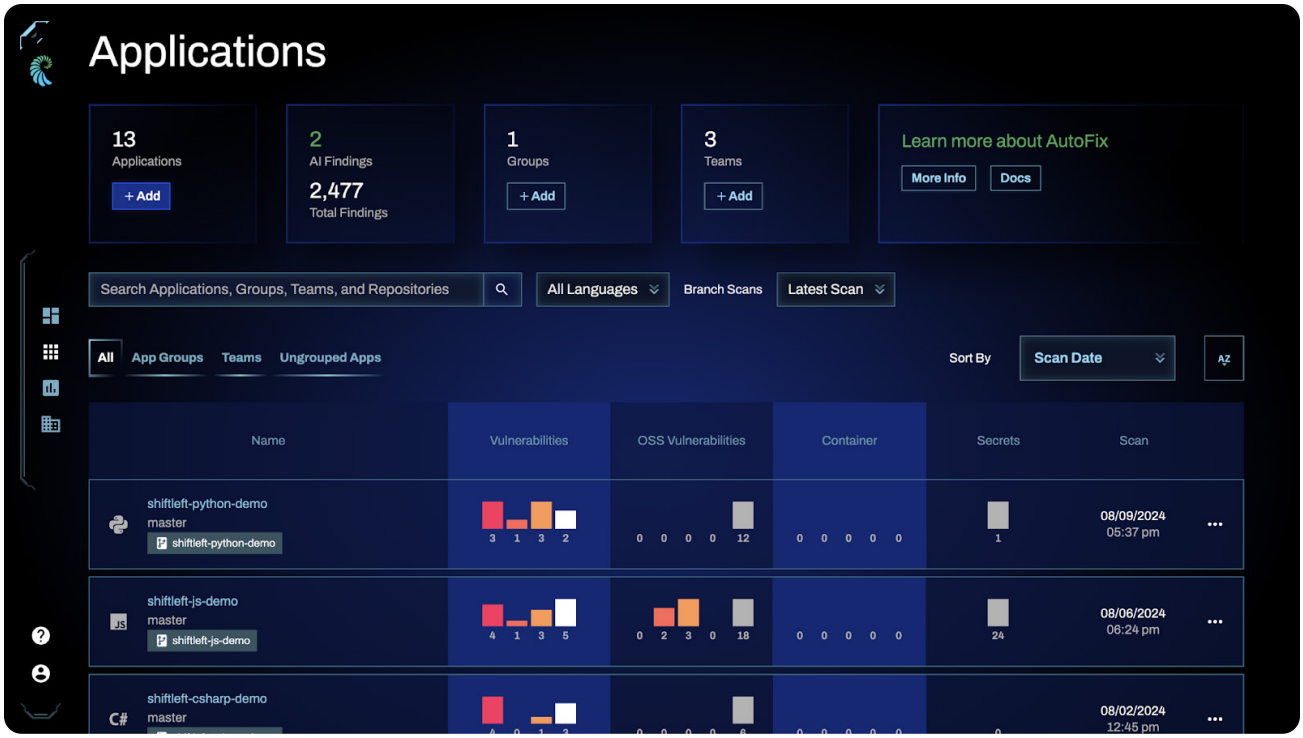


- **AutoFix Acceleration and MTTR Reduction:** While the PoC primarily focused on detection and prioritization, Qwiet AI's AutoFix was also evaluated on select high-priority issues. The results were impressive – in cases where AutoFix was applied, the time from detection to having a validated fix available shrank dramatically. For example, a critical hard-coded secret finding was automatically resolved by replacing it with a secure vault reference, and the fix was delivered for review in minutes. Developers estimated this would usually take a day to coordinate and fix properly. By using AutoFix on even a subset of issues, the **mean-time-to-resolution (MTTR) for those vulnerabilities dropped by an order of magnitude**. The PoC showed that broad utilization of AutoFix could decrease the average MTTR for vulnerabilities by approximately 30-40%, leading to quicker closure of security tickets. This results in an improved security posture and a reduced exposure window for the enterprise.

- **ROI and Developer Cost Savings:** The combined effect of the above factors (less noise, smarter focus, faster fixes) has a direct economic impact. After extrapolating the PoC results to an enterprise-scale codebase, the organization saw the potential of saving ***thousands of hours of developer time per year***. For instance, if 1,860 security findings were generally reported in the production code (as an estimate), legacy processes might require addressing all of them at great expense. With Qwiet AI, if only ~10% (186) are truly actionable, the effort drops drastically. A simplified ROI calculation showed that **developer remediation hours could be reduced by roughly 90%**, translating to hundreds of thousands of dollars saved in labor at an average developer rate of $100/hour. One scenario projected over $330,000 in savings on remediation efforts using reachability filtering and automated fixes on a large codebase. While results vary by environment, the direction is clear: Qwiet AI improves security outcomes cost-efficiently by letting teams "do more with less." The reduced toil also has an intangible ROI in developer satisfaction, engineers spend less time on tedious vulnerability backlogs and more on building product features.

- **Quality and Compliance Gains:** Another outcome noted was improved quality of fixes and easier compliance reporting. Because Qwiet AI's findings and fixes come with rich context (e.g., proof-of-exploit and compliance mappings), the security team found it easier to demonstrate compliance with internal policies and external standards. During the PoC, they mapped Qwiet's output to OWASP Top 10 categories and found comprehensive coverage. Also, the fact that suggested fixes were aligned with OWASP and NIST guidelines meant fewer reworks; security architects did not have to reject fixes for not meeting policy, since the AI already followed best practices. This built confidence that Qwiet AI could help the organization maintain standards like **OWASP ASVS** requirements and even supply evidence for audits (ISO 27001 clauses or NIST 800-53 controls related to secure software development).

In short, the real-world trial of Qwiet AI preZero underscored the platform's ability to elevate an organization's security posture while streamlining workflow. Intelligently filtering and automating AppSec tasks **multiplies the effectiveness of the AppSec team**. As one might put it, it's like adding an army of tireless junior security engineers who work at computer speed, guided by the expertise of senior security researchers encoded into the system. This case study illustrates that such a force multiplier can lead to tangible ROI in terms of time and cost savings, all while reducing risk exposure.



Comprehensive Security Coverage: Beyond Just Code Scanning

While Agentic AI, CPG, and AutoFix are at the heart of Qwiet AI's value proposition, an enterprise solution must cover more than source code vulnerabilities. Qwiet AI's platform is built as a comprehensive AppSec solution for modern development pipelines, integrating multiple layers of security scanning and risk management into a unified experience. It supports the following key areas:

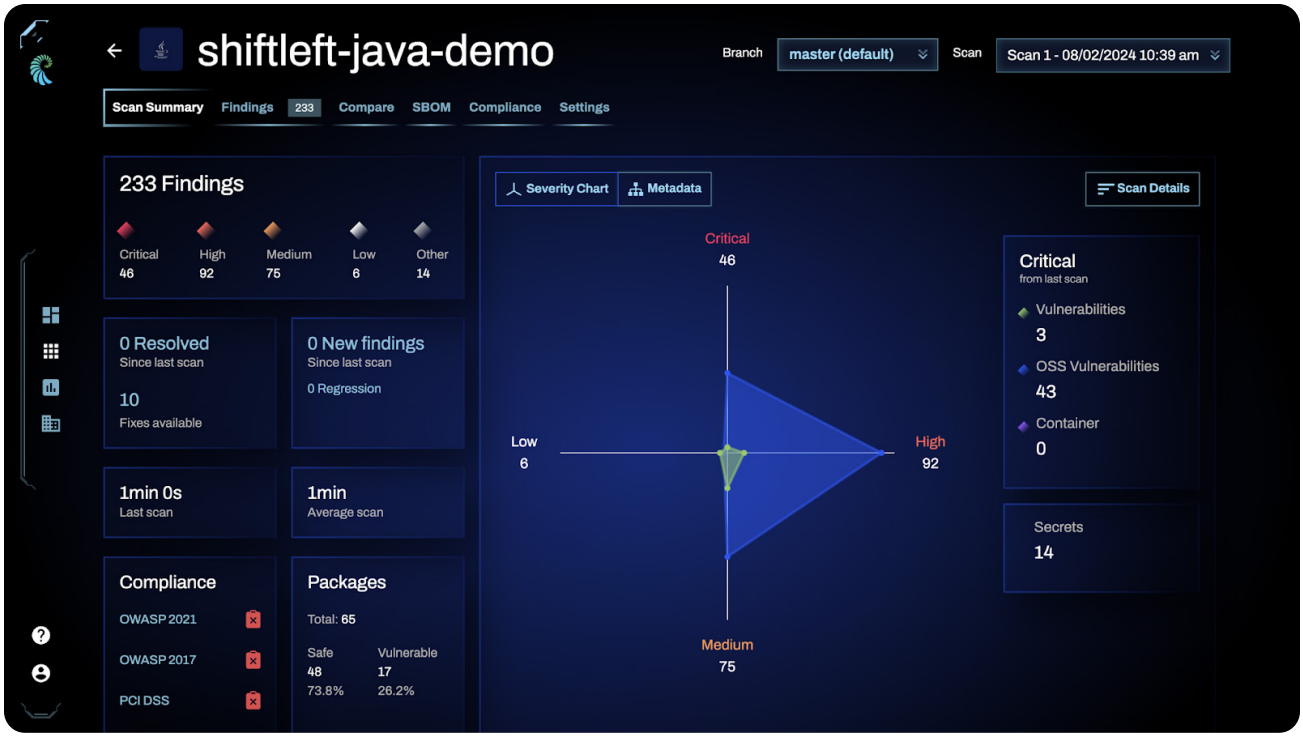
- **Static Application Security Testing (SAST):** Scans proprietary application code (across languages like Java, Python, JavaScript, C#, etc.) for security vulnerabilities such as injections, broken access control, insecure design, etc. Using the CPG, Qwiet's SAST finds issues with high accuracy and context. It detects the OWASP Top 10 and beyond, uncovering common flaws and complex logical vulnerabilities. The unified approach ensures **deeper, more accurate vulnerability detection** than legacy SAST, with fewer false alarms.
- **Software Composition Analysis (SCA):** Inventories open-source libraries and third-party components to identify known vulnerabilities (CVEs) and license risks. Qwiet AI's SCA is enhanced with the CPG context; it doesn't just list vulnerable packages, but tells you if that vulnerable code is invoked in your application. Prioritization is crucial: if your code doesn't call a library's vulnerable function, the vulnerability can be safely deprioritized. Conversely, if a high-severity CVE is in a reachable path, Qwiet flags it as a priority. The platform automatically generates a **Software Bill of Materials (SBOM)** for your application and containers, providing transparency into all components. This SBOM is continuously analyzed for new vulnerabilities, helping organizations stay on top of supply chain risks.
- **Infrastructure as Code (IaC) Scanning:** Analyzes cloud and infrastructure configuration code (like Terraform, CloudFormation, YAML, etc.) for misconfigurations and insecure settings. Qwiet AI checks these against known security benchmarks (AWS Well-Architected, CIS benchmarks, etc.) and flags issues such as open security groups, overly permissive IAM roles, or unencrypted storage. By integrating IaC scanning, preZero helps catch cloud configuration issues before they go live. This ensures that cloud and

container infrastructure are thoroughly scanned to prevent problems like a misconfigured container or Kubernetes setting from being exploited. The platform's policy engine allows IaC issues to be mapped to compliance requirements (like PCI DSS cloud controls or ISO 27001 Annex A controls), enabling proactive governance and audit readiness.

- **Container and Image Security:** Qwiet AI's platform incorporates container scanning by default, analyzing container images (e.g., Docker images) for known vulnerabilities in bundled libraries and system packages. This includes scanning for issues in base operating system layers and open-source dependencies, such as outdated OpenSSL versions or vulnerable libraries embedded in the container. Unlike other solutions that require separate licensing for container security, Qwiet AI includes this functionality without additional setup or cost. While configuration analysis (e.g., Dockerfile misconfigurations) is not currently included, container scanning results are correlated with SCA and SAST findings to provide a comprehensive view of application-layer risk at release time.
- **Secrets Detection:** The platform can detect hardcoded secrets, passwords, API keys, and other sensitive tokens in code repositories. These are critical to catch early, as exposed credentials can lead to severe breaches. Qwiet's AI can find such secrets and suggest remediation steps (e.g., rotating the credential, using a vault, and expunging it from git history).
- **Continuous Integration/DevOps Integration:** The Qwiet AI platform integrates with source control (e.g., GitHub), CI/CD pipelines (e.g., Jenkins), and IDEs. This allows scanning to be triggered by pull requests, with results commented directly on the PR, or run as a quality gate in CI. For developers, issues can be surfaced directly in their IDE as they code, thanks to agent-based analysis and pattern recognition running in the background. The platform can enforce preventative measures, such as blocking a merge due to a high-risk SQL injection, which adds a safety net to the development cycle. Additionally, by integrating with production monitoring tools (e.g., SIEMs, runtime detectors), the platform can extend this safety net into production. Any fix can have an associated monitoring rule to catch similar issues at runtime, effectively bridging DevSecOps with SecOps.

All these capabilities are accessible through a unified reporting and analysis interface that gives security and IT leaders deep visibility into their application security posture. From SAST and SCA to container scans and SBOM generation, the platform provides a trusted source of truth for tracking vulnerabilities, monitoring remediation metrics like MTTR, and maintaining compliance. It consolidates the most critical insights into an actionable format across engineering, security, and governance teams. They can generate compliance reports showing SAST, SCA, container scan results, track the MTTR for vulnerabilities, and maintain an inventory (SBOM) of all software components.

The breadth of coverage (SAST, SCA, IaC, containers, SBOM, etc.) means Qwiet AI's preZero can replace or consolidate several point tools. This lowers expenses and tool fatigue, allowing the agentic AI and AutoFix to use information across different areas. For example, suppose a vulnerable library is detected through SCA. In that case, the system can determine if the library is being used (via CPG) and then potentially create a patch (or suggest a safe upgrade) automatically for that specific use case. Or if an IaC scan finds a storage bucket misconfigured, an agent could propose the correct secure configuration. It's a comprehensive approach where **code, configuration, and dependencies are all secured in context**.



Alignment with Industry Standards and Best Practices

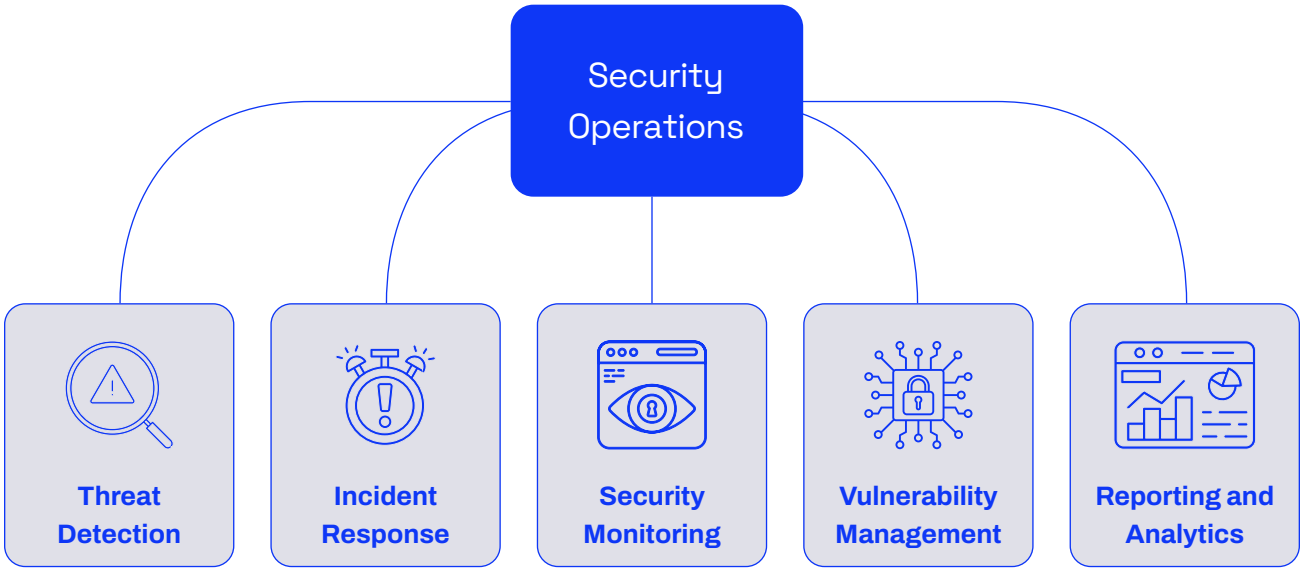
In application security, aligning with well-known standards and frameworks is crucial for credibility and compliance. Qwiet AI's platform has been built with these standards in mind, ensuring that its findings and fixes are not only compelling but also **industry-aligned:**

- **OWASP Top 10 and OWASP Best Practices:** The OWASP Top 10 is a widely recognized list of the most critical web application security risks. Qwiet AI's detection capabilities comprehensively cover vulnerabilities in the OWASP Top 10 (e.g., injections, XSS, authentication issues, sensitive data exposure, etc.), and its remediation suggestions directly tie to OWASP-recommended fixes. For example, if a finding relates to "Injection" (OWASP A03:2021), the AutoFix will likely implement input validation or use of prepared statements as per OWASP guidance. The platform's built-in rules and the training of its AI agents incorporate the **OWASP Secure Coding Practices** guidelines. The platform's output allows developers to benefit from OWASP best practices, even if they are unfamiliar with the recommendations. Additionally, reports can be filtered or summarized by OWASP category, which is proper for compliance and tracking improvements against these known risk areas.

- **NIST Frameworks and Guidelines:** Many organizations follow NIST guidelines, such as the NIST Cybersecurity Framework (CSF) or NIST Special Publications (like SP 800-53 for controls, SP 800-218 for secure software development). Qwiet AI helps fulfill several aspects of these. For instance, NIST SP 800-218 (Secure Software Development Framework) calls for organizations to implement tool-based code scanning and automated testing of security requirements – Qwiet AI's SAST and automated testing via agentic workflow directly support this. The **Agentic AI validation loop** can be seen as implementing NIST's recommended "verify the integrity of security fixes" control. Moreover, by logging all actions and providing evidence of vulnerability mitigation, Qwiet AI facilitates the reporting needed for NIST CSF's Detect and Respond functions. If mapped to NIST 800-53 control families, preZero contributes to SI

(System and Information Integrity) controls by automating flaw remediation and SA (Systems and Services Acquisition) controls by ensuring code is reviewed/scanned. Essentially, Qwiet AI operationalizes many NIST guidelines by embedding them in the development lifecycle.

- **ISO 27001 Compliance:** ISO 27001 is an information security management standard that, among other things, requires organizations to manage vulnerabilities in systems and applications (Annex A.14 - secure system engineering and A.12 – vulnerability management are relevant). Organizations can use Qwiet AI to demonstrate a code vulnerability detection and remediation process that addresses these ISO controls. The "audit-ready" nature of Qwiet's fixes means that for each vulnerability, there is an audit trail of how it was discovered and resolved, which is invaluable evidence during ISO 27001 audits or audits for other standards like SOC 2. Furthermore, the platform's operations in a secure cloud with strict data handling can help in fulfilling ISO 27001 requirements around protecting sensitive data during testing. Qwiet AI can be part of an ISO 27001-compliant secure development lifecycle, enforcing policy (like coding standards) and catching deviations.
- **Other Standards and Initiatives:** Qwiet AI's focus on automating secure development aligns with emerging initiatives like **BSIMM (Building Security In Maturity Model)** and **OWASP SAMM (Software Assurance Maturity Model)**, which emphasize shifting security left and integrating security into development. Integrating AI-driven security feedback and fixes into developer workflows enhances an organization's security maturity. Incorporating IaC and container scanning aligns with industry-standard security frameworks like CIS Benchmarks and the AWS Well-Architected Framework for cloud-native applications.

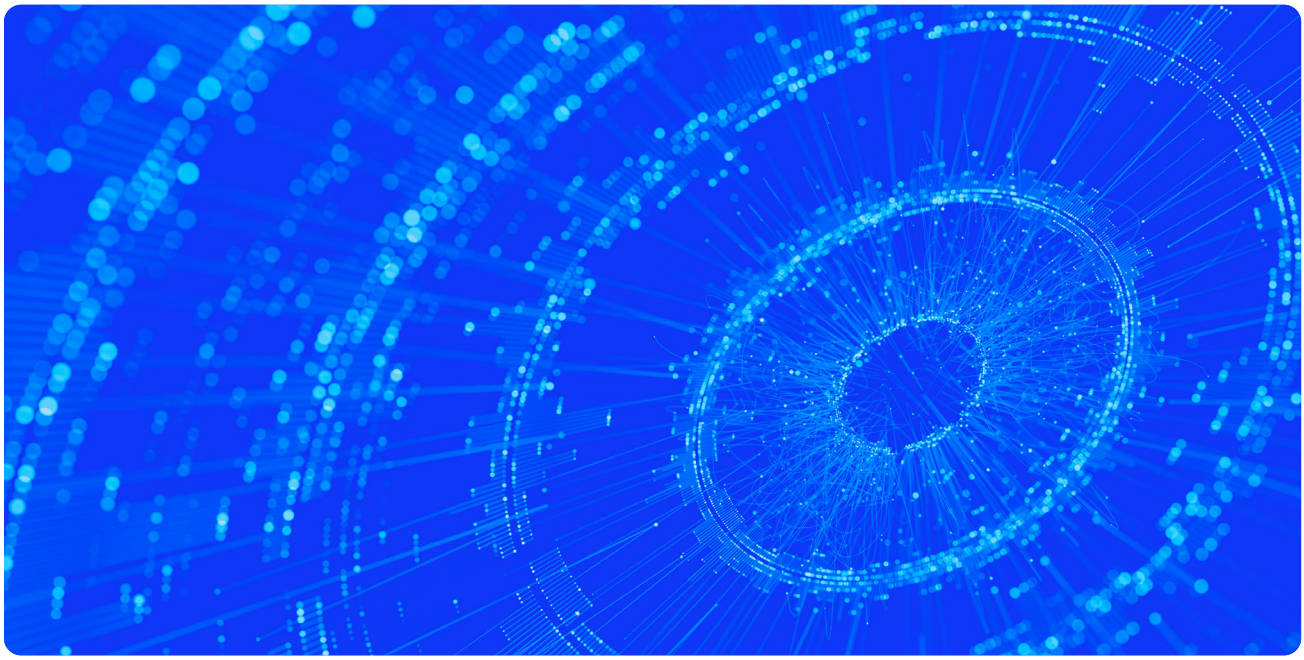


In practical terms, security leaders can use Qwiet AI's outputs to cross-reference against compliance checklists. For example, if a policy says "No high severity findings in the OWASP Top 10 categories shall go to production," Qwiet's dashboard can show if any such findings remain open. Or if ISO 27001 control A.12.6.1 (management of technical vulnerabilities) requires timely action on vulnerabilities, the platform's metrics on time-to-fix can demonstrate compliance, especially when AutoFix is handling issues rapidly.

By weaving standard references into its ruleset and output, Qwiet AI ensures that adopting the platform doesn't just solve technical problems but also helps satisfy governance requirements. This dual focus on **security and**

compliance means stakeholders like CISOs and IT auditors see Qwiet AI not as a black box, but as a tool that enforces the same principles they trust. For instance, when AutoFix applies a change, it might tag the fix with references (like “Mitigates OWASP A1 - Injection”), which provides clarity and confidence.

In summary, Qwiet AI’s preZero solution is built to ***do the right thing*** regarding security and in the ***right way*** according to industry consensus. This alignment reduces friction in adoption – teams can be assured that they are following proven security practices – and it eases the task of meeting formal security requirements and passing audits.



Conclusion

Qwiet AI’s next generation AI-native approach revolutionizes secure software development, achieving autonomous application security by combining Agentic AI with our Code Property Graph and AutoFix capabilities. This technology streamlines the security lifecycle into a continuous, AI-driven loop, maintaining quality and context in the SDLC.

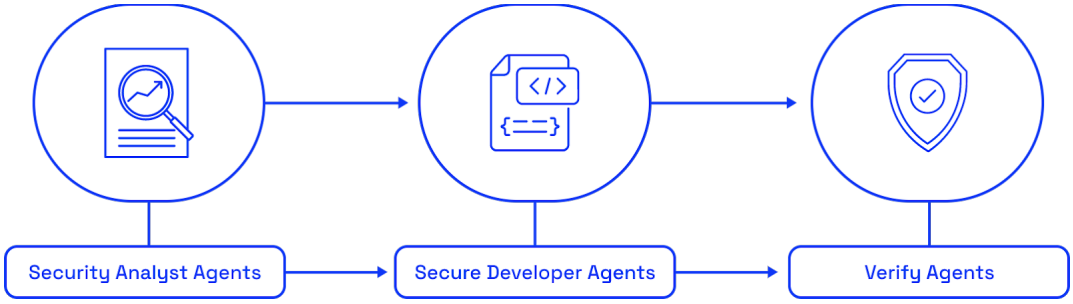
For security executives and enterprise architects, Qwiet AI offers solutions to scale AppSec or CodeSec without linearly increasing staff and technical burden. The multi-agent approach assists operations with guided expertise from numerous scenarios; addressing the challenge of resource limitations, triaging, and vulnerability management. The benefits include faster fixes, reduced labor costs, and improved risk posture.

Qwiet AI preZero acts as a force multiplier for AppSec and CodeSec engineers by automating repetitive tasks such as false positive triage and writing patches, allowing engineers to focus on higher-level concerns such as strategic decision-making, architecture, and security policy. The AI's explainability and audit trails provide transparency and build trust, while seamless IDE and pipeline integration with automated fixes and pull requests foster a DevSecOps culture and shift security left.

Technically, Qwiet AI's approach uses graphs, AI, and automation to overcome the limitations of traditional tools, providing actionable findings and correct fixes at high speed and scale. The platform's support for SAST, SCA, IaC, container security, and SBOM generation makes it a comprehensive AppSec solution.

The growing complexity of software and the evolving nature of threats necessitate an AI-driven approach that can adapt and learn while providing a comprehensive view. Qwiet AI preZero represents such an approach, moving organizations toward the goal of self-healing software, where code can identify and rectify issues under appropriate guidance. By significantly decreasing the number and duration of vulnerabilities, the combination of agentic AI and CPG in Qwiet AI preZero brings us closer to that goal. Adopting Qwiet AI's preZero means embracing a future where security is autonomous, continuous, and deeply integrated into the development lifecycle, enabling development and security teams to work together with AI to achieve accelerated innovation and security.

The difference between staying ahead of cyber threats and innovating quickly could be the alignment of development and security teams, which could be the key to staying ahead of cyber threats and innovating quickly. As demonstrated in this whitepaper, the technology is not theoretical, it's here today, delivering measurable improvements. Organizations that leverage these advances can transform their AppSec programs from a bottleneck into a competitive advantage, shipping software faster and safer by design.



Sources

Qwiet AI Official Website

Qwiet AI AutoFix Overview

Qwiet AI CPG Technology

Eng et al., 2024 – Multi-Agent Chain-of-Thought for Bug Fixing
Reference discussed in multiple Qwiet documents

OWASP Secure Coding Practices

OWASP Top Ten 2021

NIST SP 800-218 (SSDF)

NIST SP 800-53 Rev. 5

ISO/IEC 27001 Overview

Forbes Tech Council – DevSecOps Adoption Insights

Ready to secure your codebase with Agentic AI?

Request a personalized demo to see how Qwiet AI delivers faster fixes, fewer false positives, and more intelligent security workflows powered by Code Property Graphs and AI AutoFix.

Request a Demo

qwiet AI