Sonar

The State of Code

Volume 2: Security

Table of Contents

Introduction 3

Overview and summary of key findings 4

Our methodology 5

Code security findings 6

Spotlight: Secrets 10

Conclusion 11

About Sonar 11

Introduction

Sonar's integrated solutions for code security and code quality, called SonarQube, examine 300 billion lines of code every single day in order to help development teams ship high-quality, secure code. Given our unique visibility into the code developers are creating today, we analyzed a subset of that code in order to extract the top issues that surfaced through our analysis. This is the second in a series of reports highlighting our findings.

In this report, we highlight the most common code security issues that can affect software quality. This context is essential for software developers, their leaders, and AppSec stakeholders who need to make (or justify) software engineering investment decisions around training, tooling, or technical debt and would benefit from knowing what issues may be lurking or unknown in their critical software. And as AI coding assistants generate more code, the quality of existing application source code becomes more important, as it is the main data used to train these AI tools.

For background, Sonar measures the impact of code issues in every project or codebase across three software qualities: reliability, security, and maintainability. These three areas are deeply interconnected in high-quality code: poorly maintained code typically develops reliability issues and security vulnerabilities over time. Taken together, reliability, security, and maintainability determine not just the initial success of software but its long-term value, adaptability, and total cost of software ownership throughout its lifecycle. This is why every rule violation identified by Sonar is automatically assigned an impact quality for one of these three areas.

As mentioned, we focus primarily on the security issues in this report. Our prior report, "The State of Code: Reliability," is also worth a read. Stay tuned for future reports, where we'll explore maintainability issues (sometimes called code smells) as well as common issues broken down by programming language.

Overview and summary of key findings

Software plays a vital role in modern business, increasingly serving as a key differentiator and driver of customer experience. Consequently, the quality of internally developed software will have a growing impact on both customer satisfaction and competitive standing.

Poor software quality, therefore, represents a significant threat to every business. Recent projections estimate that the annual cost of poor software quality in the US has risen to over $2.41 trillion (Consortium for Information & Software Quality, 2022).

The staggering growth of new code that is generated or touched by AI tools dramatically increases the impact of this trend. AI coding tools are excellent mimics: they create based on what they learn from existing human code. It follows that code quality problems that commonly exist today will continue to pop up in AI generated code, and these issues must be addressed.

This report includes the most common security issues impacting our users' software. Sonar's analysis exposed security issues, like log injection vulnerabilities and hard-coded secrets (such as passwords, keys, and credentials), that could cause severe consequences for applications running in production if not addressed.

Key findings:

- On average, every million lines of code Sonar analyzed contained about 1,200 security issues (vulnerabilities and security hotspots requiring review).
- Said differently, Sonar caught 1-2 security issues per developer per month during the period examined.
- The most frequently found security vulnerabilities relate to log injection attacks.
- The most frequently found security hotspots relate to hard-coded credentials and IP addresses. Hard-coded credentials was also the most frequently surfaced blocker issue relating to security.
- 50% of the secrets found by Sonar were associated with database passwords.

Our methodology

Sonar operates the largest SaaS solution for integrated code security and code quality analysis, so we have a unique view into how code is being written across many languages, industries, organization sizes, and geographies. We looked at Sonar analysis data for the last six months of 2024, across seven of the most common languages developers use. Unlike other reports that rely on surveys, this analysis is backed by concrete data that shows the real issues developers are encountering as they code. Moreover, each of these data points links back to an issue that was caught by Sonar and surfaced to developers.

About our dataset

The dataset for this report comes from SonarQube Cloud, which is a SaaS code analysis solution designed to detect coding issues in over 35 languages, frameworks, and Infrastructure-as-Code (IaC) platforms. It integrates directly with continuous integration (CI/CD) pipelines and DevOps platforms (like GitHub, GitLab, Azure DevOps, and Bitbucket), and checks code against an extensive set of rules covering maintainability, reliability, and security issues on each merge/pull request or branch commit. Our analysis evaluates code health against thousands of criteria that have been created by developers, for developers.

For this study, we've included every pull request and branch analysis in 2024 between July 1st and December 31st, where the code was written in **Java, JavaScript, TypeScript, Python, C#, C++, or PHP** (the top software development languages used with SonarQube).

This scope yields a vast dataset encompassing:

- More than **7.9 billion lines of code**.
- Work from over **970,000 developers** across over **40,000 organizations** globally.
- Roughly **445 million code issues** across **5,300 unique quality and security rules**.

Code security findings

Sonar flags security issues as either vulnerabilities or security hotspots. A **vulnerability** indicates that the code could be open to attacks and requires immediate action, while **security hotspots** are security-sensitive pieces of code that need manual review to determine if they pose a threat. Sonar also detects many types of **secrets**, which could grant unauthorized access to secure systems and data if not resolved.

The most common security vulnerabilities

Of the approximately 445 million issues identified by Sonar, 1.3 million were categorized as security vulnerabilities. This translates to roughly 170 vulnerabilities per million lines of code analyzed.

All of the security vulnerabilities identified below are tracked by MITRE in the Common Weakness Enumeration (CWE).

Logging should not be vulnerable to injection attacks

Languages: C++ Java PHP Python **Volume:** ▢▢▢▢▢▢▢▢

Log injection happens when an application doesn't sanitize untrusted data used for logging. Attackers can forge log content to hide malicious activities, compromising the integrity of the log. Developers should ensure data used for logging is content-restricted and typed by validating or sanitizing the content.

Exceptions should not be thrown from servlet methods

Languages: Java **Volume:** ▢▢▢

Servlets process HTTP requests and generate responses, and use exceptions to manage unexpected errors. Surrounding method calls with try/catch blocks is crucial, because uncaught exceptions can lead to denial-of-service attacks, unintended application states, or exposure of sensitive data in stack traces.

Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks

Languages: C++ Java JS TS PHP Python **Volume:** ▢▢

Cross-site scripting (XSS) attacks permit malicious code to be injected into web applications when user input is not properly encoded before being included in HTTP responses. To prevent XSS attacks, developers should encode user input using secure libraries and ensure that template engines are configured to automatically encode output.

Database queries should not be vulnerable to injection attacks

Languages:     **Volume:** ☐

Database injections (e.g., SQL injections) occur when an application uses unsanitized data in a database query, allowing attackers to modify the query logic maliciously. This can lead to identity spoofing, data manipulation, data deletion, and remote code execution. To fix this, use prepared statements to compile the query logic and placeholders before receiving user data, effectively sanitizing the input.

Weak SSL/TLS protocols should not be used

Languages:    **Volume:** ☐

Older versions of SSL and TLS are considered weak and officially deprecated. Attackers can recover plaintext from encrypted data by performing cryptographic attacks. Developers should use cryptographic algorithms that are considered strong by the cryptographic community.

Why you should watch out for log injection attacks

Log injection vulnerabilities compromise the integrity of system logs that security teams depend on for monitoring and investigation. When attackers inject malicious content through unsanitized input, they can effectively rewrite the historical record—hiding their activities, inserting false information, or framing innocent users. This prevents breach detection, misleads incident response teams, and obscures attack patterns, giving attackers more time to operate undetected. Without trustworthy logs, organizations lose a critical security tool, potentially allowing attackers to compromise systems while leaving little evidence of their intrusions.

The most common security hotspots

A security hotspot is a piece of code that a developer should review as it may be a security risk. It does not necessarily indicate a vulnerability, but flags areas that could be sensitive and require further inspection to ensure code security. Fixing security hotspots adds protection against potential threats: the more hotspots that are reviewed and addressed, the more secure the application becomes against attacks. Reviewing security hotspots helps developers understand and address the specific risks that could arise within their code.

Of the approximately 445 million issues in Sonar's dataset, 8.4 million were categorized as security hotspots. This translates to roughly 1,100 hotspots per million lines of code analyzed.

Hard-coded credentials are security-sensitive

Languages: C# JS TS php Python **Volume:** □□□□□

Credential leaks often occur when a sensitive piece of authentication data is stored with the source code of an application. These credentials should be revoked immediately, and a secret vault should be used to generate and store a replacement.

Using hard-coded IP addresses is security-sensitive

Languages: Python TS JS C# php Java C++ **Volume:** □□□

Hardcoding IP addresses in source code can cause issues in product development, delivery, deployment, and security. Developers should use environment variables or configuration files for IP addresses instead, or use a domain name if confidentiality is not required.

Using slow regular expressions is security-sensitive

Languages: JS TS Java Python **Volume:** □□□

Regular expressions can cause performance issues due to backtracking. To avoid this, ensure your regex doesn't have problematic repetitions, consecutive repetitions that can match the same content, or unbounded repetitions in partial matches.

Delivering code in production with debug features activated is security-sensitive

Languages: Java Python C# **Volume:** □□

Debugging features in development tools and frameworks should never be enabled in production. They can leak sensitive system information and pose a security risk. Developers should ensure these features are never enabled in production environments or applications distributed to end users.

Not specifying a timeout for regular expressions is security-sensitive

Languages:  **Volume:** ☐☐

Regular expressions without timeouts, especially those processing untrusted input with patterns vulnerable to catastrophic backtracking, can lead to Denial-of-Service attacks. Developers should always specify a matchTimeout and review the patterns to ensure they are not vulnerable.

Why you should avoid hard-coding credentials

Hard-coded credentials create severe security risks because they could become accessible to anyone with repository access and remain preserved in version control history even after removal. This practice enables uncontrolled secret sharing, creates credentials that rarely expire, and typically means all environments share the same sensitive data. When code repositories are exposed through leaks or breaches (which happens frequently), these hard-coded secrets provide attackers with direct access to critical systems. Additionally, embedding secrets bypasses proper security controls like auditing, rotation policies, and the principle of least privilege, creating long-lived vulnerabilities that are difficult to track and remediate. Fortunately, it's quite simple to detect secrets in code from within the IDE (using something like SonarQube for IDE) to catch and prevent this kind of leak.

Spotlight: Secrets

In addition to the static code analysis this report has explored thus far, Sonar also has the ability to detect hard-coded secrets in source code. SonarQube's secret detection is designed to protect source code from exposing sensitive information, including passwords, API keys, encryption keys, tokens, and database credentials. It uses a combination of regular expressions and semantic analysis to detect secrets at the earliest stage of development within the IDE, as well as in CI/CD pipelines.

We've extracted this SonarQube's detection data for secrets in the period between October 1—December 31, 2024. Within this time period, over 465,000 hard-coded secrets were detected, the vast majority of which are considered blocker issues. Developers should strongly consider revoking any secrets detected to be hard coded in software, and issue new ones that are stored securely in a secrets vault.

50% of the secrets found by Sonar in the source code scanned in this dataset were associated with database passwords.

Database passwords should not be disclosed

Leaked database passwords can have serious consequences, including compromise of sensitive data, reputational damage, and security downgrades for applications that rely on it.

Azure Storage Account Keys should not be disclosed

Azure Storage Account Keys authenticate access to Azure Storage resources. Leaking this key can allow attackers to access and modify your data, create or delete resources, and incur charges on your account.

MongoDB database passwords should not be disclosed

MongoDB passwords authenticate database users and grant data access permissions. Leaked passwords can compromise sensitive data, cause reputational damage, and downgrade application security.

Amazon Web Services credentials should not be disclosed

Leaked AWS secrets can allow attackers to take control of AWS infrastructure. This can result in infrastructure takeover, DNS redirection, malicious instances, DDoS attacks, and more. Attackers could also leverage cloud infrastructure as a gateway to other assets, causing further damage.

OVH keys should not be disclosed

Leaked OVH secrets can allow attackers to take control of OVH infrastructure. This can result in infrastructure takeover, DNS redirection, malicious instances, DDoS attacks, and more. Attackers could also leverage cloud infrastructure as a gateway to other assets, causing further damage.

Conclusion

This report is intended to be a first step towards increased transparency around some of the most common security issues that can be found in the source code being actively written and maintained today. It underscores the need for solutions that facilitate automated code review, like SonarQube, to intercept critical issues so they don't escape into production environments.

As our community moves increasingly towards using AI to augment software development and dramatically increase the rate at which new code is created, we think this assessment of the state of code they are analyzing with Sonar will help to highlight frequently-occurring potential failure points and help developers strengthen the value of the code they create.

About Sonar

Sonar helps developers accelerate productivity, improves code security and code quality, and supports organizations in meeting compliance requirements while embracing AI technologies. The SonarQube platform, used by 7M+ developers worldwide, analyzes all code – developer-written, AI-generated, and third-party open source code – supercharging developers to build better applications, faster.

Sonar provides code review and assurance, inherently applies secure-by-design principles, fixes issues in code before they become a problem, and enforces policy standards – all while improving the developer experience. Sonar is trusted by the world's most innovative companies and is considered the industry standard for integrated code quality and code security. Today, Sonar is used by 400K organizations, including the DoD, Microsoft, NASA, Mastercard, Siemens, and T-Mobile.

Trusted by over 7M developers and 400K organizations

Microsoft NASA NVIDIA Pfizer Mercedes-Benz

AIRFRANCE BARCLAYS COSTCO WHOLESALE Johnson&Johnson Santander

Learn more at sonar.com