apriorit

CASE STUDY

Implementing Cross-Platform SBOM for a Cybersecurity Product

CYBERSECURITY

Background

A global cybersecurity software vendor requested Apriorit's assistance with expert-level development tasks for a cross-platform security solution.

As their product grew in both stack and functionality, they found it increasingly difficult to efficiently manage third-party dependencies and emerging vulnerabilities.

The product had to meet strict compliance requirements, as it serves enterprise customers and organizations operating in regulated industries. And with changes introduced by laws and regulations like the Cybersecurity Resilience Act (CRA) and Digital Operational Resilience Act (DORA), automated dependency management had become a necessity.

The client wanted to get ahead of legal/regulatory and buyer expectations before SBOM requirements became blockers in procurement, and they entrusted the Apriorit team with strategizing and executing this task.

The client

Our client is a software vendor who delivers cybersecurity solutions to enterprise and public-sector organizations worldwide. Their customer base includes organizations from highly regulated industries, such as healthcare and finance, for whom security vetting of third-party software components is a standard part of procurement.

Our client:
Cybersecurity software provider

Operation geography
Global

Industry:
Cybersecurity

Collaboration with Apriorit:
Continuing partnership

Solution we delivered:
- SBOM integration
- Automated vulnerability management pipeline

Services we provided:
- Custom software development
- DevSecOps
- CI/CD integration
- Cybersecurity engineering

The challenge

When our client approached us with this request, they needed expert assistance with two tasks:

1) Find a way to automate dependency management across a highly versatile technology stack.
2) Change the vulnerability management workflow from reactive to proactive.

Our client's solution worked with several technology stacks that required different tooling:

- C++ code for Windows, Linux, and macOS
- C# code for the server
- TypeScript/JavaScript code for the front end

Given this, there was no single scanning approach that would work across the board. Yet the scanning results had to feed into a unified view.

On top of that, the client's vulnerability management was mostly reactive.

Sometimes, issues surfaced through customer-run audits rather than through internal processes. As a result, the client's team had to address them during active procurement or post-delivery.

The client wanted to make their vulnerability management less reactive, enabling earlier detection and mitigation.

Project details

To implement SBOM integration across all technology stacks, the team worked with the following tools and technologies:

SBOM standard

CycloneDX

Vulnerability management

Dependency-Track

CI/CD platform

TeamCity

Issue tracking

Jira

Identity management

Active Directory

Target technology stacks

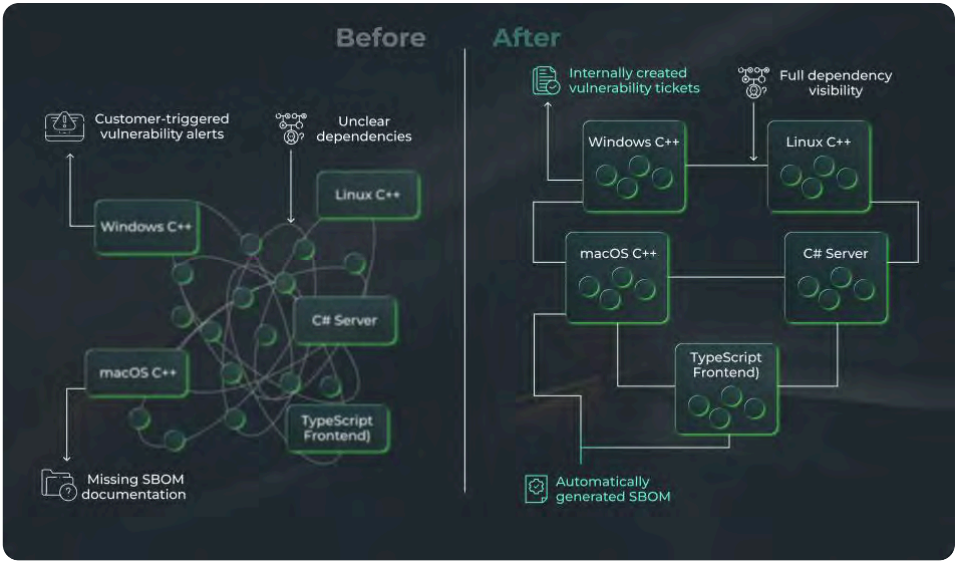
- Windows C++
- Linux C++
- macOS C++
- C# (.NET) (server side)
- TypeScript/JavaScript (front end)

The result

After handling this multi-level task, the Apriorit team delivered:

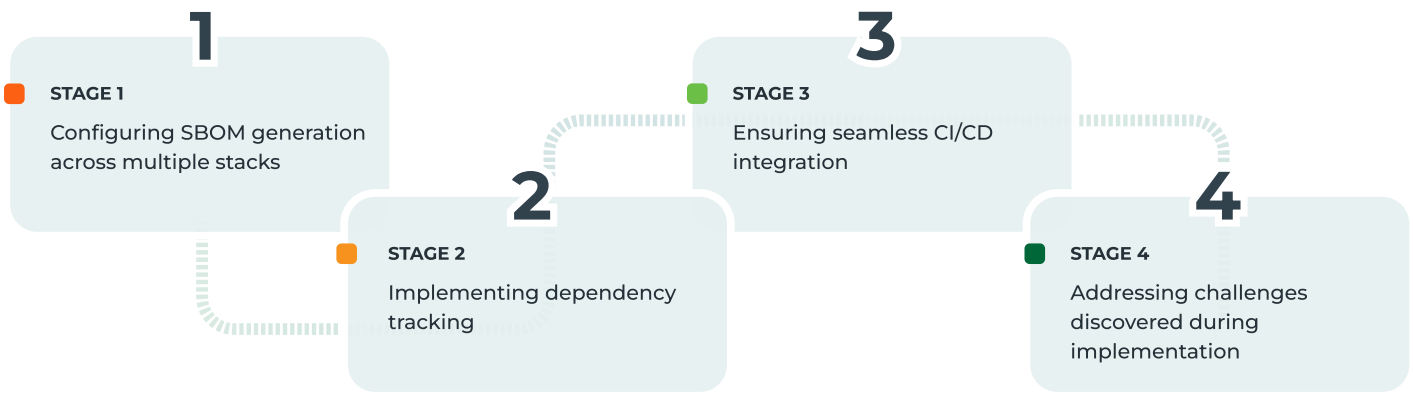
- Automated SBOM generation across all technology stacks, integrated into the CI/CD pipeline as weekly builds
- Centralized vulnerability tracking with automatic Jira issue creation for newly discovered CVEs
- On-demand compliance reports and SBOM exports in required formats for enterprise customers
- A functional strategy for addressing detected vulnerabilities depending on the impact of fixes

Impact of SBOM and vulnerability management integration by Apriorit:



How we did it

While planning SBOM implementation, the Apriorit team focused on four core areas:



1. Configuring SBOM generation across multiple stacks

The first decision was tooling. We selected:

- Dependency-Track (DT) as the centralized management platform
- CycloneDX as the SBOM standard

At the time of implementation in early 2025, DT was the only viable free solution offering a centralized portfolio view with CI integration. Alternatives provided scanning capabilities but lacked a unified storage or reporting layer.

At the time, CycloneDX tooling was more mature than its SPDX equivalents, reducing implementation risks.

Next, we addressed the per-stack scanning challenge.

Each technology stack required a dedicated tool configuration to generate a valid CycloneDX SBOM. We identified and configured the appropriate scanner for each, then set up the pipeline to forward all generated files to Dependency-Track for consolidation.

2. Implementing dependency tracking

Deploying DT was straightforward — the more important work was seamlessly integrating it with the client's existing development ecosystem.

First, we configured Active Directory authentication, keeping access management consistent with the client's existing identity infrastructure.

Then our team configured email reporting for vulnerability summaries and set up Jira integration so that newly detected vulnerabilities automatically generated issues. This part required us to carefully set up all user permissions and issue templates. It took several iterations to get the template structure right for the client's existing Jira workflow.

As a result, DT behaved as a native part of the client's development environment.

3. Ensuring seamless CI/CD integration

At this stage, we configured SBOM generation and vulnerability scan sessions.

Build times for SBOM jobs ranged from one to twenty minutes, depending on the subproject. Since this didn't impact the primary development pipeline, there was no need to change anything here.

Instead, our team focused on improving the SBOM scanning script.

Sometimes, DT didn't see the existing project file, and sometimes, components got duplicated during scans.

We consolidated component versions so that no components with different versions would be duplicated within the same subproduct. This mostly happened because parts of the dead code in the client's product sometimes referred to older component versions.

Given that third-party components don't change on every commit, a daily cadence was sufficient. Running SBOM-related scans on every build would have added overhead without meaningful benefit.

We decided to run vulnerability scanning only on release branches for the current and previous versions and to exclude individual feature branches. This approach kept integration essentially invisible to day-to-day development while ensuring the team had up-to-date vulnerability data for everything that mattered.

To maintain clear project versioning in DT, we used release names instead of project build numbers, preventing excessive SBOM versions in the system.

Addressing challenges discovered during implementation

While not an initially planned focus area, this became an important part of the project.

- **Many vulnerabilities after the first scan.** This was an expected result for a first-time vulnerability detection scan. However, it was important to carefully structure and prioritize the backlog to avoid overwhelming the team.

  We reviewed the initial report, filtered out false positives, and created a component update plan to address confirmed vulnerabilities. Going forward, when a scan detects new vulnerabilities, the system automatically generates a Jira ticket and adds it to the development backlog, assigning a priority based on the risk score — the same as for any other project task.

- **False positives in DT scans.** The component matching mechanism in DT-based scanning relies on package identifiers (Package URL and Common Platform Enumeration) and often produces false positives. If, say, a scanner assigns an incorrect identifier or the vulnerability database entry contains errors, there will be a mismatch. For example, a C++ library vulnerability may be incorrectly flagged in the C# codebase due to a partial name match via CPE. We had to mark such cases as false positives directly in DT and set the corresponding Jira issue to the dedicated Won't Fix status.

- **Inability to fully implement a zero-tolerance approach.** Our team commonly adopts this approach as the steady-state policy. Usually, we process and address all confirmed vulnerabilities, regardless of their risk score. However, fixing some detected vulnerabilities would require migrating to an incompatible library version. In those cases, we agreed that the team would perform an exploitability analysis before deciding on a path forward.

Impact

With SBOM integration in place, the client is now more compliant with legal and regulatory requirements, such as the EU Cyber Resilience Act and DORA. The client's development team now has full visibility into third-party dependencies across the entire product, with all components, versions, licenses, and known vulnerabilities consolidated in a single dashboard.

The most significant operational change is the shift from reactive to proactive vulnerability and dependency management.

> In our field, customers commonly run component-level security audits before signing a contract, so an unmanaged software supply chain is a direct commercial liability.
>
> Automating SBOM generation and vulnerability detection across all our technology stacks enables our team to identify and address dependency risks long before our customers even notice them.
>
> VP of Engineering,
> Cybersecurity software provider

Vulnerabilities that previously surfaced during customer security audits now appear as Jira tickets, which are addressed through the normal development process. The team no longer has to handle dependency issues that were first discovered only during procurement.

For enterprise customers, the improvement is equally concrete: Compliance reports on vulnerabilities and dependencies, as well as SBOM exports, are available on demand in the formats chosen by end users. What previously required days of manual preparation and was error-prone is now automatically generated in minutes.

The Apriorit team continues to assist our client with this project, helping their team improve performance and harden product security.