Martina Latoria
ENG 298
8 December 2025

Module 7 Report

Introduction

In this lab, I analyzed the software supply chain of the NG911 application using two different SBOM tools: Syft and Trivy. The goal was to understand how SBOMs support secure design principles like transparency, open design, and defense in depth. After generating the SBOMs, I used Grype to identify vulnerabilities (CVEs) and compare how different SBOM formats represent components. This entire process reinforced how important visibility and verification are when maintaining secure software.

Part 1: SBOM Generation

1. Syft SPDX SBOM

I generated the first SBOM using Syft:
syft . -o spdx-json > ../deliverables/sbom_syft_spdx.json

Syft reported 107 components in the NG911 repository.
The output followed the SPDX JSON format, which is structured, detailed, and focuses on package metadata such as:

- package name
- Version
- Checksum
- License
- supplier information (if available)

The SPDX structure is more formal and is often used in compliance or audit processes.

2. Trivy CycloneDX SBOM

Next, I used Trivy to generate a CycloneDX SBOM:
trivy fs . --format cyclonedx --output
../deliverables/sbom_trivy_cdx.json

The CycloneDX SBOM contained 1,976 components, which is significantly higher than Syft. Trivy detects lower-level artifacts (like metadata files, configs, and language-specific modules), which explains the bigger component count.

CycloneDX focuses on dependency relationships and is more commonly used in application security workflows because it tracks how components relate to each other.

3. Comparison of SBOM Formats

Aspect	**Syft (SPDX)**	**Trivy (CycloneDX)**

Component Count	107	1976
Focus	Compliance & metadata	Dependency graphs & Security
Structure	Strict, standardized JSON	Flexible format with security fields
Level of Detail	Higher-level packages	Much deeper enumeration

The biggest difference was how many components each tool considered “significant.” Syft focused on main packages, while Trivy included nearly every artifact related to dependencies.

Part 2: Vulnerability Analysis
I scanned the Syft SBOM using Grype:
grype sbom:../deliverables/sbom_syft_spdx.json -o table > ../deliverables/vuln_analysis_grype.txt

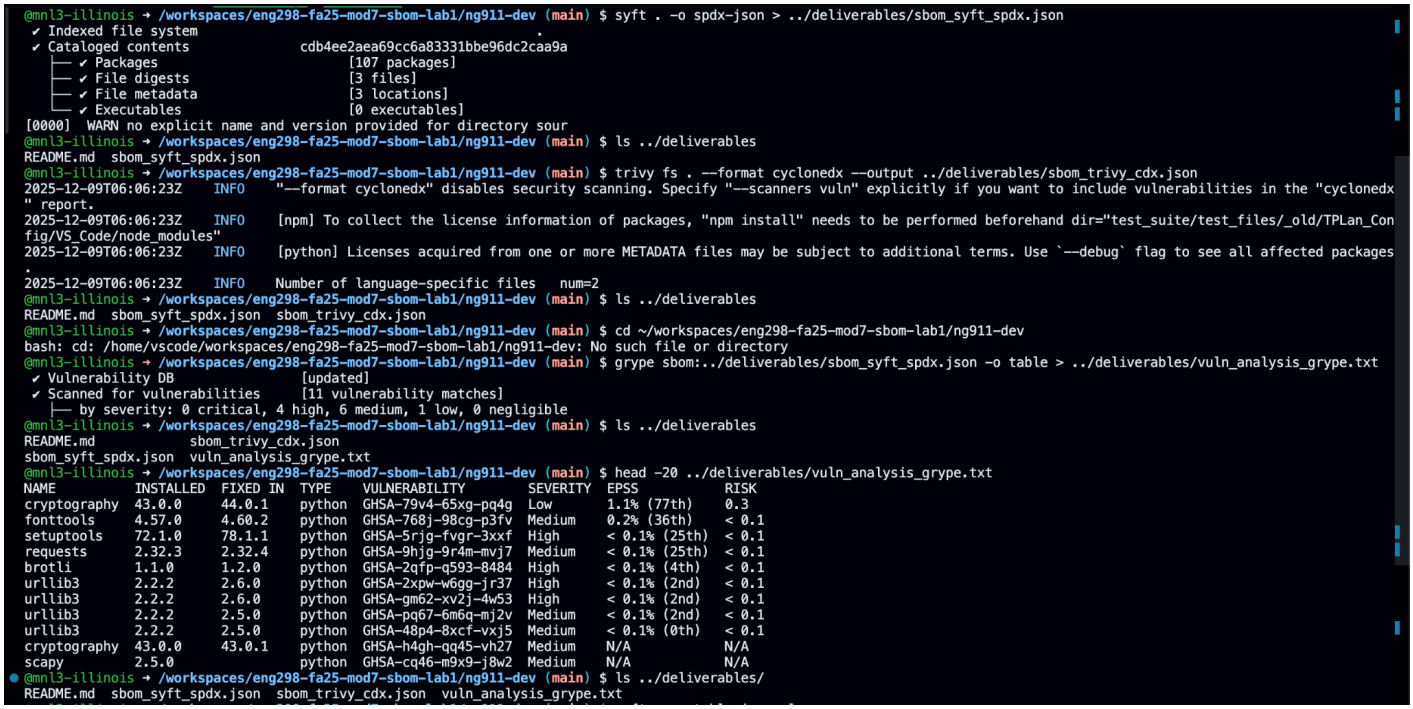
^ This produced a list of vulnerabilities across several Python libraries in the NG911 repository. Below are the Top 5 CVEs from the output:

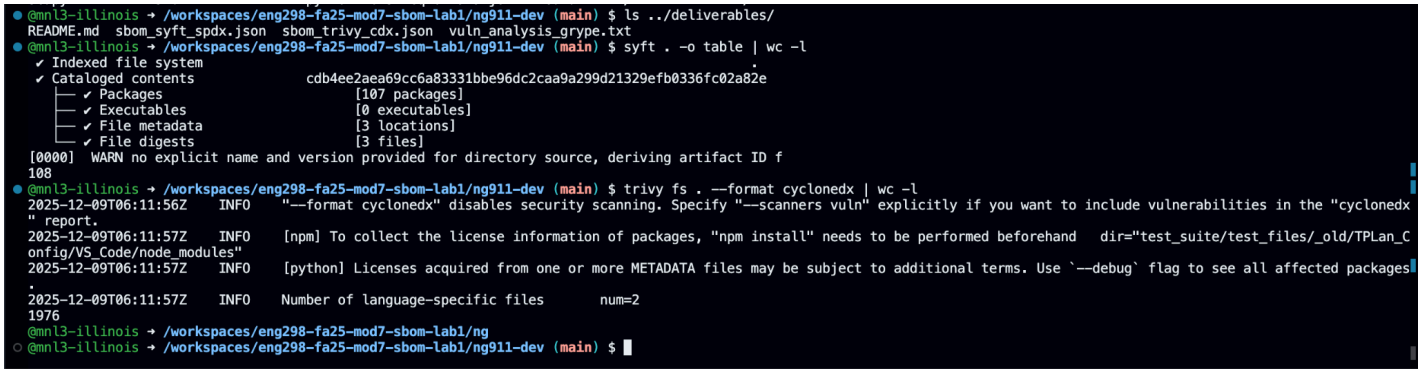
CVE/GHSA ID	Severity	Component	Version	Comment
GHSA-79v4-65xg-pq4g	Low	cryptography	43.0.0	Minor cryptography issue; fixed in 44.0.1
GHSA-768j-98cq-p3fv	Medium	fonttools	4.57.0	Could impact font parsing functionality
GHSA-5rjg-fvgr-3xxf	High	setuptools	72.1.0	High-severity packaging vulnerability
GHSA-9hjg-9r4m-mvj7	Medium	requests	2.32.3	Possible flaws in request handling
GHSA-2qfp-q593-8484	High	brotli	1.1.0	Compression vulnerability; upgrade recommended

I choose GHSA-5rjg-fvgr-3xxf, which affects *setuptools*.

According to the NVD, this issue can allow malicious code execution during package installation because certain metadata fields are not properly validated. This is a big deal because setuptools is widely used for Python packaging, so a compromised or tampered package could affect the entire build environment.

Screenshots:





Reflection:

This lab honestly helped me understand why SBOMs matter way more than I thought. When I ran Syft and Trivy, I was surprised by how many components the NG911 repo actually had. Syft only showed around a hundred, but Trivy found almost two thousand. I definitely wouldn't have known any of that without using these tools. It made me realize how much of modern software is built on top of other libraries and dependencies you don't normally see.

The two SBOM formats also felt really different in practice. SPDX (from Syft) was cleaner and more straightforward, while CycloneDX (from Trivy) dug way deeper and listed almost

everything. After comparing them side by side, it made sense why teams might use both depending on what they care about: SPDX for simple inventory, and CycloneDX for more detailed security-related info.

Running Grype on the SBOM also showed how many vulnerabilities can show up even in pretty normal Python packages, like requests or setuptools. That part made the most sense to me: if you don't even know what's inside your codebase, you can't possibly know what's vulnerable or needs updating.

Overall, this lab helped me see how SBOMs, scanning, and verification connect to the bigger security ideas we learned about, like visibility, complete mediation, and defense in depth. It's basically about knowing what you have, checking it, and then making sure nothing slips through the cracks. I feel like I actually get why SBOMs are such a big deal now, especially when new CVEs come out all the time.