Tracking Dependencies and Security Risks in the Maven Architecture using Neo4j and Goblin Weaver Proposal

By Daniel Pang, Ahmed El Shatshat

The Problem

Software security will always and forever be more and more important and dependency management is crucial in maintaining secure software. The question becomes, how can we leverage dependency management to better improve a system's security? Taking inspiration from the many papers discussed in class as well as background research pertaining to software architecture, one can see that how a software project is structured, and as such how dependencies are formed in the software, can directly impact how quick the software lifecycle can be and as such, how quickly security concerns can be addressed. Given that Maven has been around over 20 years, one can expect changes in software architecture or simply changes in dependency structures over its lifetime, providing a viewpoint on the relationship between artifact dependency structure and security risks.

The Data and Tools

Fortunately, all the information that should be required for this line of research is included in the Goblin framework as is, including the additional Weaver metrics used to enrich the Maven Central dependency graph.

Simply, all that is required is the Neo4J database of the enriched Maven Central dependency graph and the Goblin Weaver REST API. This is as is, provided by the MSR 2025 Mining Challenge.

Research Questions

There are three research questions attached to this project.
RQ1: Has Maven trended towards software architecture that has fewer dependencies over the years?
RQ2: If Maven has trended towards fewer dependencies, has this resulted in faster software lifecycles?
RQ3: Has faster software lifecycles reduced security risks in Maven?

How to do the Work

Intentionally, the research questions have a natural order to them, that is, the former precedes the latter. First, artifacts must be found that have made notable reductions in the number of dependencies over the history of the project. Following that, upon finding said artifacts, find their number of releases to see if these artifacts have a faster software release cycle, this will most likely take the form of a median of freshness. Finally, for these artifacts with faster release cycles, compare how quickly their responses are to CVEs in comparison to other CVEs in other artifacts.

For a rough schedule of milestones.
M1: Identification of appropriate artifacts that have potential to answer our research questions (large changes in dependency, security adjacency)
M2: Extraction of relevant artifacts, and assessment that they are indeed relevant to our research
M3: Analysis of artifacts and trends that can be found therein
M4: Analysis of security risks of artifacts that have had large dependency changes
M5: Compilation of research to form conclusions, writing of paper

Threats to Validity

There are a few threats to validity that can be expected even at this stage. Firstly, without delving deep into the code base itself, it is difficult to prove that a reduction in dependencies is a result of a refactoring or change in software architecture. Secondly, different CVEs will naturally have different complexities in how to handle them and as such a faster response to a CVE may not be a result of a shorter software life cycle but could be a myriad of other factors such as the CVE itself being an easier issue to handle.

There are also a few potential problems that can be mitigated through an adjustment in research criteria. Firstly, if there are no or few artifacts with notable reductions in dependencies over time, an alternative approach would be to attempt to compare artifacts by their number of dependencies such that one would expect a trend that artifacts with fewer dependencies would have shorter software lifecycles. Secondly, following that, if there are no such artifacts with having shorter software lifecycles over the lifetime of the project, it would not be possible to see if CVE response times have improved. In response to this possibility, another solution would be to attempt to correlate CVE response times to software lifecycles in Maven Central as a whole.