Homework 1 Report

Robert Wallace – rwallac1@nd.edu

Design Decisions

I use the NVD REST API to extract information from NVD, using the requests library. I did not end up using an API key, though I do have one, because I never got rate limited and so did not notice that I wasn't using it in the first place. I added it in the end to make sure that the TA can easily add the API key

To parse the pom files, I used the lxml library to parse the pom files and navigated and to find the dependencies after removing comments.

To match CVE information to dependencies in a pom file, I select all configurations and relevant columns from each row where the configurations is LIKE the artifactID and either the descriptions or the reference is LIKE the groupId's vendor property. Under this, this should find the vulnerabilities that reference the dependency in question, though it may not cover vulnerabilities in that dependency, since those will not mention the groupId, much less be guaranteed to have a similar artifactId

My database is simply a 263,161-row table containing every CVE from 1999 to 2024, last updated September 17, 2024, 7:42 PM, with columns as shown in Figure 1

Knowledge Base Statistics

For example: "The database includes the tables shown in Figure 1. It contains 1 table, containing all the data that I thought relevant when first importing the data. Further revisions would clean this up and remove unused data, though it may be useful for automatically updating or similar purposes.

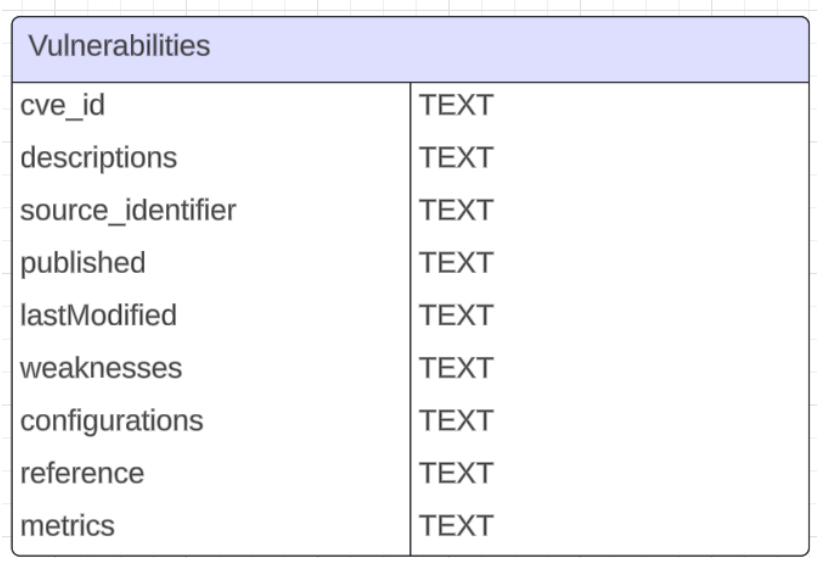

Figure 1

Research Synthesis

1. **What is the motivation and problem being tackled by this paper and how they solve this problem?**

The motivation and problem being tackled by this paper is the ever-increasing number of Open-source vulnerabilities along with the cost of detecting and fixing these. Additionally, merely knowing that there is an exploit may not be enough information. Is this exploit likely to affect a particular application? Or is it a niche case that is not likely to cause immediate harm, while another exploit that may appear less dangerous at first may be more exploitable and therefore immediately problematic. This tool solves this by focusing on a code-centric rather that metadata-based analysis, to determine the reachability of the vulnerable portion of that library.

2. Given the ideas explored in the paper above, how could you change your vulnerable dependency finder to incorporate some ideas described in it?

Their method of searching for vulnerable constructs independently of library would be a useful tool to add as well as the NVD known vulnerability search, precisely because of the prevalence of reused code in OSS software. Dynamic analysis would be done after determining that there is a vulnerable dependency in the current program, to see if a particular discovered vulnerability is likely to be reachable during actual execution. If this does not find any likelihood of being executed, the user would still be warned about the vulnerability, but also informed that at the

moment there is no major risk of that section of the dependency being run. The idea of implementing a metric for the stability of the project and developer effort required in the case of a dependency update would be implemented after the analysis of current vulnerabilities, to give the developer a better understanding of which vulnerabilities would be easily mitigated.