Vulnerabilities in Open-Source Language Servers

Henry Post

Intro to OS

1. Introduction

Open-source software use is widespread, and powers most of modern software development. However, open-source software is not always thoroughly tested, and can contain security vulnerabilities. The tools that most software developers use to write code use a component called a “language server”, and most language servers are open-source software themselves. However, not all of these language servers are guaranteed to be free of vulnerabilities. In this paper, I aim to sample a few of the most popular language servers and run security testing tools against them, to determine how vulnerable they may be.

1.05 Organization

This paper is broken up into different parts, and I will outline them sequentially. The introduction section is meant to give you a sense of why this paper is being written, and what problem as well as what problem domain it seeks to solve. The problem domain section is meant to give you more context on why this is a problem, and what benefits may be achieved as a result of this paper being completed. The specific problem section is meant to present the core problem of this paper. The hypothesis section is meant to present my hypothesis to you. The related research section presents related research. The metrics section presents metrics and charts that are related to my research to you. There is a subsection called “Metric Conclusions” that outlines the conclusions that I draw from each metric. The possible future work section is a list of ideas for future papers. The conclusion section contains conclusions that I draw from completing this paper, and whether or not my hypothesis was correct. Finally, the citations section is just a list of citations.

1.1. Problem Domain

Language servers are necessary in modern software development, and assist software developers in writing code. They provide auto-completion of code, allow jumping to function definitions and documentation, and can provide error checking and type checking to a code editor. In this paper, I will examine various open-source language servers for security vulnerabilities, using different open-source SAST (Static application security testing) tools, and will compare the security results and draw conclusions from them.

Language servers, by virtue of their place in-between programming languages, open-source packages, and development environments, present a unique attack vector that if exploited, could lead to cascading effects on the applications being developed. A compromised language server could introduce unintended behaviors or malicious code into software projects.

Open-source software is still very susceptible to security vulnerabilities. Having many contributors with varying levels of skill and reputability contribute to open-source software projects can speed up development and foster innovation and rapid feature addition, but it can also introduce flaws into software projects. This paper’s focus on securing widely-used pieces of software development tooling is not just an academic exercise, but also a necessary step in securing development tooling that is used by many software developers to write software that is used widely across the world.

2. Specific Problem

The use of language servers within IDEs has grown in recent years. Language servers are a bridge between programming languages, open-source libraries, and development environments. As these pieces of software are widely-used, it's important to make sure they're secure. The core problem this paper aims to address is the potential security vulnerabilities present in open-source language servers. A compromised language server has the potential not only to disrupt individual developers, but also to spread malicious code throughout any software that is created with its help. Given how widespread language servers are, even a small vulnerability could affect thousands of software projects.

3. Hypothesis

I hypothesize that open-source language servers, due to their wide number of contributing developers, may possess significant security vulnerabilities. By employing free SAST and SCA tools to analyze different language servers, I anticipate finding differences in the security posture of these servers. I also posit that any found vulnerabilities are likely due to the nature of open-source contributions to software, where contributors with varying levels of skill and intent can introduce unintentional or even malicious flaws.

4. Related Research

[1] This paper outlines a method by which an attacker could stealthily introduce vulnerabilities into an open-source project. This is a useful paper, as it outlines the necessity of performing automated security scans on open-source projects.

[2] This paper is a survey of vulnerabilities, and their associated fixes, in open-source software. About 12% of the vulnerabilities discovered had no CVE, and were likely not publicly known. This is an important paper, as it outlines the potential for undiscovered vulnerabilities within OSS libraries.

[3] This paper outlines a technique for better assessing the impact of vulnerabilities that are discovered. This is accomplished by attaching additional metadata to the vulnerability, and correlating it with how the open-source software is used by the code that depends upon it.

5. Metrics

5.10 Language Servers surveyed

Below is a list of surveyed language servers. 17 different language servers were chosen, and they vary in their language, star count, and fork count.

Lan-guage Server Name	Git URL	Forked Git URL	Stars	Forks	Language	Imple-menta-tion Lan-guage
ccls	https://github.com/MaskRay/ccls	https://github.com/in-tro-to-os-language-	3500	273	C/C++/Obj-C	C++

		server-security/ccls				
clangd	https://github.com/llvm/llvm-project/tree/main/clang-tools-extra/clangd	https://github.com/intro-to-os-language-server-security/llvm-project/	22100	8600	C++/clang	C++
Eclipse JDT Language Server	https://github.com/eclipse-jdtls/eclipse.jdt.ls	https://github.com/intro-to-os-language-server-security/eclipse.jdt.ls	1500	380	Java	Java
gopls	https://github.com/golang/tools/tree/master/gopls	https://github.com/intro-to-os-language-server-security/tools/	7000	2200	Go	Go
intelephense	https://github.com/bmewburn/vscode-intelephense	https://github.com/intro-to-os-language-server-security/vscode-intelephense	1500	86	PHP	TypeScript
Kotlin Language Server	https://github.com/fwcd/kotlin-language-server	https://github.com/intro-to-os-language-server-security/kotlin-language-server	1300	178	Kotlin	Kotlin
omnisharp-roslyn	https://github.com/OmniSharp/omnisharp-roslyn	https://github.com/intro-to-os-language-server-security/omnisharp-roslyn	1600	419	C#	C#
Phan	https://github.com/phan/phan	https://github.com/intro-to-os-language-server-security/phan	5500	372	PHP	PHP
php-language-server	https://github.com/felixfbecker/php-language-server	https://github.com/intro-to-os-language-server-security/php-language-server	1100	223	PHP	PHP
phpactor	https://github.com/phpactor/phpactor	https://github.com/intro-to-os-language-server-security/phpactor	1100	119	PHP	PHP
PHPUnit for VSCode	https://github.com/recca0120/vscode-phpunit	https://github.com/intro-to-os-language-server-security/vscode-phpunit	113	38	PHP	PHP
PowerShell Editor Services	https://github.com/PowerShell/PowerShellEditorServices	https://github.com/intro-to-os-language-server-security/PowerShellEditorServices	553	208	PowerShell	C#
Python LSP Server	https://github.com/python-lsp/python-lsp-server	https://github.com/intro-to-os-language-server-security/python-lsp-server	2600	307	Python	Python
Serenata	https://gitlab.com/Serenata/Serenata	https://github.com/intro-to-os-language-server-security/Serenata	62	?	PHP	PHP
Solargraph	https://github.com/castwide/solargraph	https://github.com/intro-to-os-language-server-	1800	148	Ruby	Ruby

		security/solargraph				
Sorbet	https://github.com/sorbet/sorbet	https://github.com/intro-to-os-language-server-security/sorbet	3500	481	C++	Ruby
TypeScript Language Server	https://github.com/typescript-language-server/typescript-language-server	https://github.com/intro-to-os-language-server-security/typescript-language-server	1500	158	TypeScript	TypeScript

5.11 Snyk SAST Vulnerabilities by category, per software

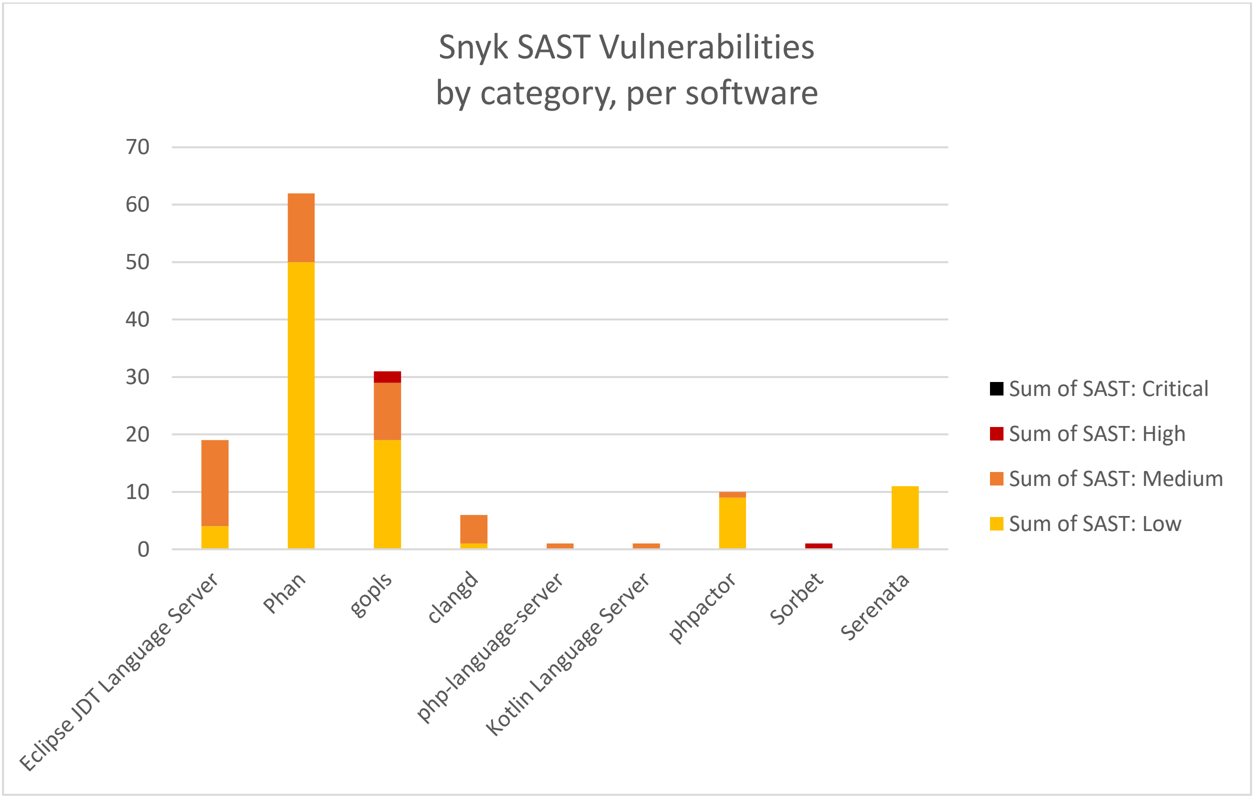

Figure 1: A chart showing a breakdown of SAST vulnerabilities by category, for each piece of software.

5.12 Snyk SAST vulnerabilities that are test code/test scripts

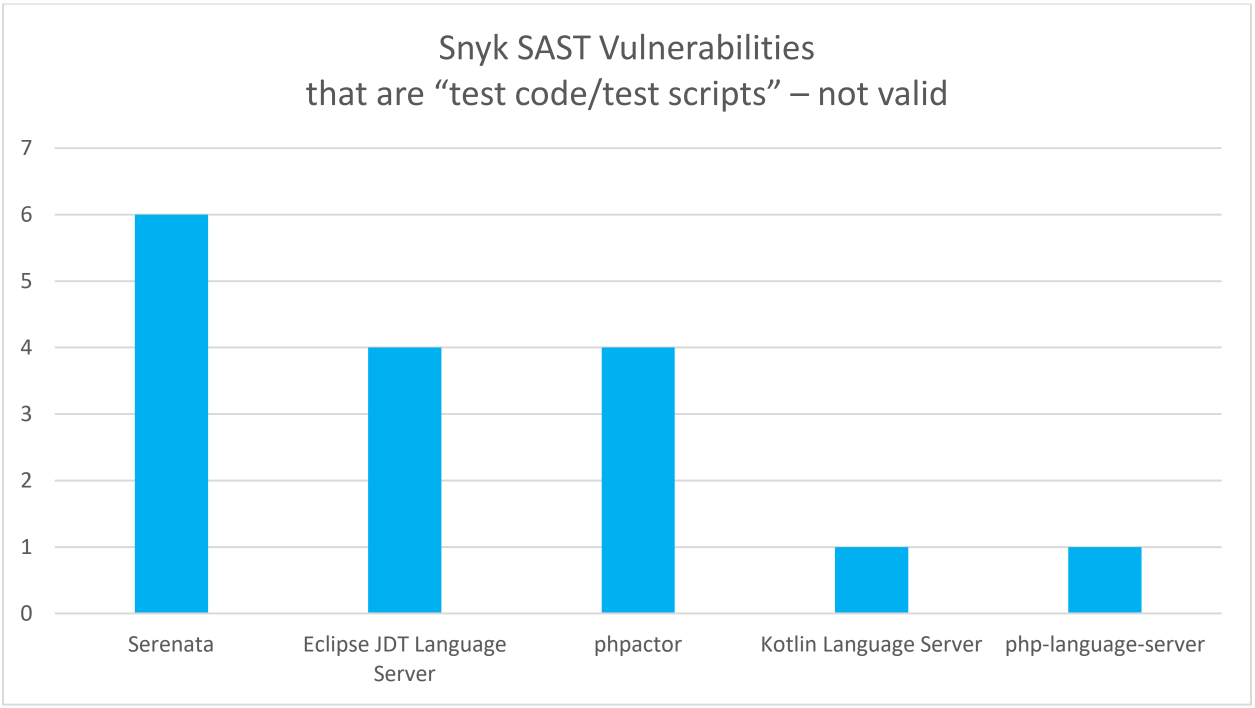

Figure 2: SAST vulnerabilities detected by Snyk that are actually test code and should not be reported. These were manually sorted by looking at the filepath of the file, or the contents of the source code file.

5.13 Snyk SCA Vulnerabilities by category, per software

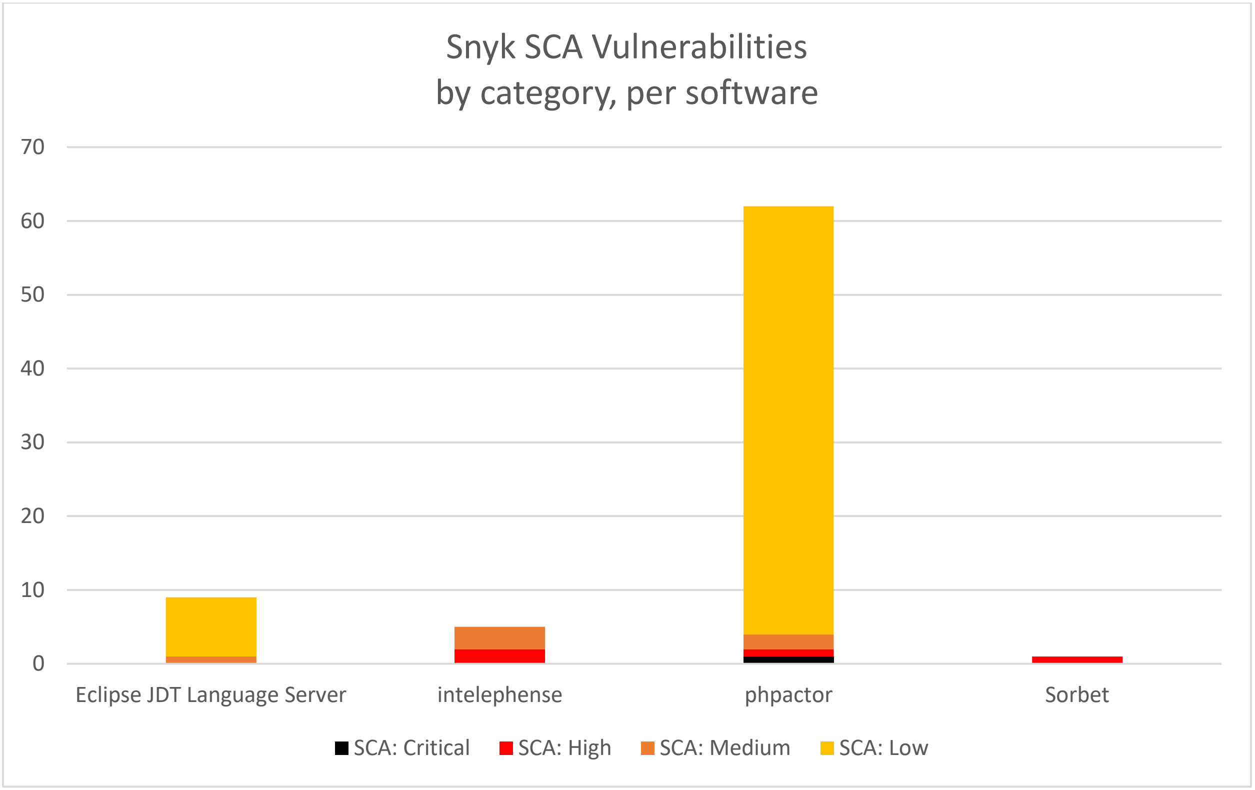

Figure 3: A list of SCA vulnerabilities by category, for each software.

5.14 Snyk SCA vulnerabilities that are test code/test scripts

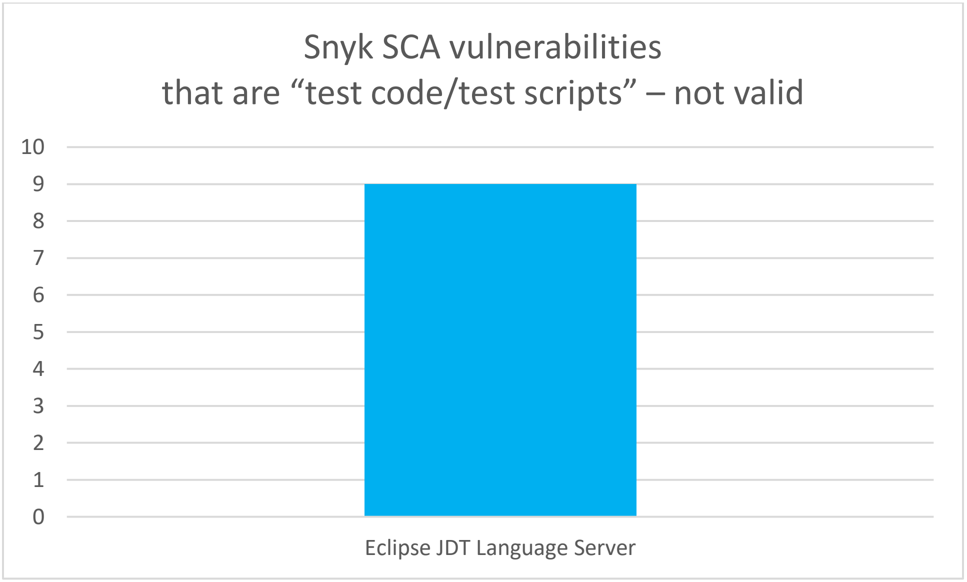

Figure 4: Snyk SCA vulnerabilities that are test code/test scripts, and not valid findings.

5.15 SonarQube SAST vulnerabilities

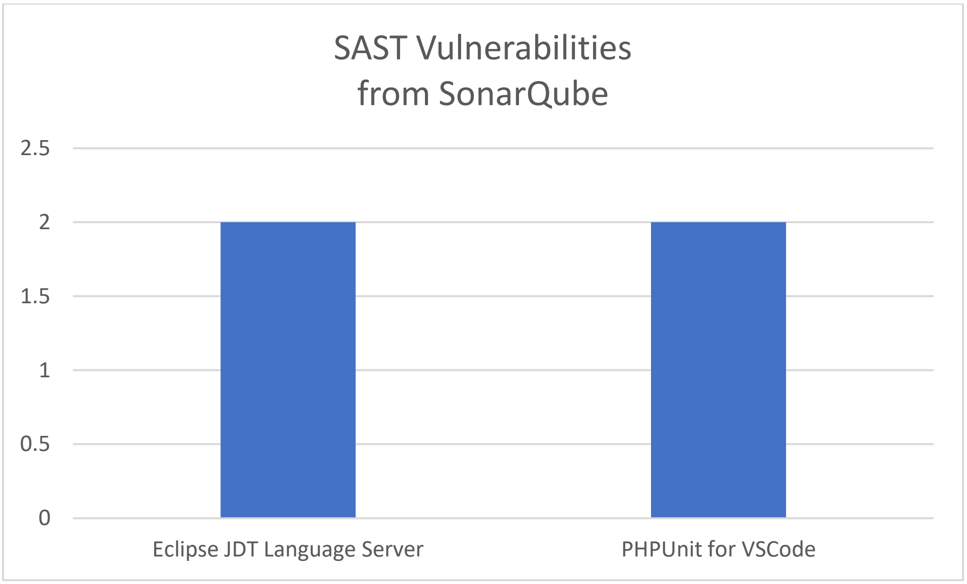

Figure 5: SAST Vulnerabilities from SonarQube, per software.

5.2 Metric Conclusions

From these metrics, I can draw a few conclusions.

The first is that many open-source projects may have significant and numerous security vulnerabilities that their authors may not know about. This supports my hypothesis.

The second is that, if one wants to do large-scale automated SAST and SCA analysis of open-source repositories, a portion of the results will likely be incorrect. 16 SAST vulnerabilities and 9 SCA vulnerabilities only showed up because test code got scanned. Out of the 142 SAST vulnerabilities and 77 SCA vulnerabilities, this represents a non-negligible "test code that got scanned but shouldn't have been" metric: (16/142) or 11.2% for SAST, and (9/77) or 11.6% for SCA.

5.3 Empirical Evidence

The metrics support my hypothesis. There are 9 projects with valid Snyk SAST vulnerabilities, 4 projects with valid Snyk SCA vulnerabilities, and 2 projects with valid SonarQube SAST vulnerabilities. Of these projects, 5 have at least 10 SAST vulnerabilities. These are non-negligible vulnerability reports, and it makes me feel that this method of testing open-source language servers should be automated in some way.

The metrics also show that, if this were to be automated in some way, there should be some auto-triaging feature that automatically excludes "test" code or deployment scripts. About 10% of the results only showed up because test code or deployment scripts got included in the scan.

6. Possible future work

Regardless of the results of this paper, I believe that all open-source projects should have automatic SAST and SCA scans performed on them. I believe that a standardized, automated system that provides high-quality security testing to all widely-used open-source projects is necessary to prevent security vulnerabilities from occurring. Software is developed too quickly and used too widely to accept any non-automated solution.

In future papers, here are some examples of potential research areas:

- An automated system for evaluating the security posture of any open-source software project
- A “software security health” scorecard system that can be used to give a score to open-source software projects
- An AI-powered system that can evaluate open-source software projects for potential security issues based on code patterns and open-source contributor behaviors
- Examine how various projects use security checks within their CI/CD pipelines
- A thorough review and comparison of various SAST and SCA tools, to determine which ones offer the most comprehensive and accurate results

7. Conclusion

In conclusion, it is necessary to scan open-source software with some form of SAST, SCA, or other automated code testing to ensure that essential code does not have vulnerabilities. Of the 17 language servers surveyed, at least 9 had vulnerabilities. Because of how widespread the usage of these pieces of software is, any vulnerabilities could show up in potentially millions of users’ code editors. Also, if automated code testing is performed, there *should* be some mechanism to automatically close issues that are only showing up because test code or deployment scripts got scanned, since these would pollute the results of potentially valid vulnerability detections.

8. Citations

[1] Wu, Qiushi and Kangjie Lu. “On the Feasibility of Stealthily Introducing Vulnerabilities in Open-Source Software via Hypocrite Commits.” (2021).

[2] S. E. Ponta, H. Plate, A. Sabetta, M. Bezzi and C. Dangremont, "A Manually-Curated Dataset of Fixes to Vulnerabilities of Open-Source Software," 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 2019, pp. 383-387, doi: 10.1109/MSR.2019.00064.

[3] H. Plate, S. E. Ponta and A. Sabetta, "Impact assessment for vulnerabilities in open-source software libraries," 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), Bremen, Germany, 2015, pp. 411-420, doi: 10.1109/ICSM.2015.7332492.