Do Developers Use Static Application Security Testing (SAST) Tools Straight Out of the Box? A Large-scale Empirical Study

Gareth Bennett
Tracy Hall
Emily Winter
{g.bennett,tracy.hall,e.winter}@lancaster.ac.uk
School of Computing and Communications
University of Lancaster, Lancaster, UK

Steve Counsell
steve.counsell@brunel.ac.uk
Dept. of Computer Science
Brunel University
London, UK

Thomas Shippey
thomas.shippey@logicmonitor.com
LogicMonitor
London, UK

Abstract

Static application Security Testing (SAST) tools are an established means of detecting vulnerabilities early in development. Previous studies have reported low detection rates from SAST tools and recommend either combining SAST tools or configuring rule sets to detect more vulnerabilities. However, while previous work suggests that developers rarely combine or configure any of the Automatic Static Analysis Tools (ASATs) they use, it is currently unclear whether SAST tools are used directly "out of the box". To understand how developers use SAST tools, we performed a large-scale survey involving 1,263 developers. We pre-screened developers to establish their SAST use and found that only 20% (204/1,003) used SAST tools. Of those developers who did use SAST tools, we found a large number did not use multiple tools (59%), did not configure tools (54%) or did neither (40%). Our results suggest that more work is needed to help developers combine and configure tools, since doing so is likely to detect significantly more vulnerabilities.

Keywords

Survey, Vulnerability Detection, Static analysis

ACM Reference Format:
Gareth Bennett, Tracy Hall, Emily Winter, Steve Counsell, and Thomas Shippey. 2024. Do Developers Use Static Application Security Testing (SAST) Tools Straight Out of the Box? A Large-scale Empirical Study. In *Proceedings of the 18th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24), October 24–25, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3674805.3690750

1 Introduction

Vulnerabilities are a serious threat to a system and to its users since their exploitation have severe consequences; some examples include EternalBlue [1] and the recent Log4Shell [2]. Early detection of vulnerabilities is essential to limit exploitation opportunity. Static Application Security Testing (SAST) tools are one established means of detecting vulnerabilities. Such tools are resource inexpensive and used in many different developer environments; as such, their deployment is widespread in Open Source Software (OSS) [10, 31].

That said, SAST tools generate a large number of false positives [21], do not detect all weakness types [18, 24, 26] and miss some vulnerabilities [24]. There are known ways to improve SAST detection rates. In particular, combining multiple SAST tools has been shown to increase vulnerability detection rates [14, 18, 19, 27, 29]. More importantly, configuring a single tool's rule set, rather than using it straight "out of the box" has been shown to outperform combinations of tools [11]. Despite these benefits, previous work suggests that developers appear to use only a limited number of static analysis tools and rely on default configurations [10, 31].

It is also important to understand human factors associated with using such tools [33] and several studies have analysed the context in which static analysis tools are used by developers. However, previous work has failed to focus on SAST tools [10, 31] or performed qualitative work with a small number of developers [6]. In this emerging results paper, we conduct a survey to understand whether developers configure and combine their SAST tools, or use them straight "out of the box". We designed the survey to meet these goals and provide quantitative results to complement previous qualitative work [6]. We address the following research questions:

RQ1: How do developers use SAST tools?
We ask developers how they currently use SAST tools and compare their use with previous work suggesting that they are rarely combined or configured.
RQ2: How do developers configure SAST tools?
Static analysis tools can be configured to add, remove, edit or create new rules. We ask developers which configurations they perform on each SAST tool they use, identifying flaws in certain SAST tools.
RQ3: Why do developers configure SAST tools?
We ask developers why they configure their SAST tool to understand what they want from them; this may highlight a disconnect between the tool provider and their user base.
RQ4: What metrics are associated with developers when configuring and combining SAST tools?
We perform statistical analysis on our survey responses to discover which developer responses are associated with whether they configure or combine SAST tools. This may highlight reasons why some developers use multiple tools or, indeed, configure tools in the first instance.

ESEM '24, October 24–25, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-1047-6/24/10
https://doi.org/10.1145/3674805.3690750

Our study contributes the following:

- Results from a large scale survey involving 1,263 employed developers regarding their SAST tool use and configurations; this data is publicly available [7].
- Statistical analysis on developer responses and discussion of the results in the wider context of static analysis tools.
- Insights into SAST tools from professional developers and recommendations for future work for SAST tool providers, SAST tool users and researchers.

The remainder of the paper is organised as follows: Section 2 describes our methodology; Section 3 then presents our results. Our findings are discussed in Section 4. We then discuss related work (Section 5) before concluding in Section 6.

2 Methodology

2.1 Survey Design & Distribution

Table 1 presents the survey questions and response types (Single choice (SC), Multiple Choice (MC), and Ranking (R)). We discuss our two pre-screening questions in Section 2.3. Questions Q1-Q8 cover current interactions with SAST tools and whether developers combine or configure them (RQ1). Questions Q9-Q11 are repeated for each SAST tool developers' use and help identify which tools are being configured and the types of configurations developers make (RQ2). Questions Q12-Q14 ask developers when and why they configure tools to understand what motivates them to do so (RQ3). We also asked eight demographic questions to understand which developers configured and combined tools (RQ4) adapted from Stack Overflow's Developer Survey [3]; however, we omit these questions from the table due to space limitations.

Table 1: Survey questions and response type.

Q	Question	Type
Q0.1	Which type of vulnerability detection tools and techniques have you used to detect vulnerabilities?	MC
Q0.2	Which of these specific vulnerability detection tools have you used to detect vulnerabilities?	MC
Q1	How many years have you used SAST tools?	SC
Q2	How do you generally use SAST tools?	MC
Q3	How often do you use SAST tools to scan your code?	SC
Q4	Which of the following explains why you use SAST tools?	MC
Q5	How effective are SAST tools at detecting vulnerabilities?	SC
Q6	Which weakness types are SAST tools particularly effective at detecting?	MC
Q7	Which SAST tools do you use?	MC
Q8	Have you configured the rule set of SAST tools?	SC
Q9	Regarding the configurability of {Q7 selected tool}, select all statements that apply. I have...	MC
Q10	How easy is {Q7 answer} to configure?	SC
Q11	Which of the following best describes why you don't configure {Q7 answer}?	MC
Q12	How often do you configure the rule set for SAST tools?	SC
Q13	When specifically are you likely to configure the rule set of SAST tools?	MC
Q14	Rank the following statements as to why you configure SAST tools?	R

Conducting a pilot study can help ensure that a study has no obvious errors or omissions [22, 25, 32]. We conducted a pilot study with Software Engineering postgraduate students who had experience with SAST tools, as they represented our target population. In response to this pilot, we added an additional question (Q6). We distributed the survey on LinkedIn and various subreddits, since these sources were relevant to developers who use SAST tools and have large active communities (771,000 active members on r/cybersecurity). We added a Capctha prompt for developers recruited from these online sources to limit automated responses. To attract further participants, we also recruited from Prolific [4], a paid recruitment service connecting researchers with professionals. Paid recruitment services have gained popularity in research because they provide a large audience of professionals, a high response rate and an easy recruitment process [23, 30]. Overall, we gathered 1,263 responses from participants (less than 1% from online sources and 99% from Prolific).

2.2 Data Handling

The surveys were created using Qualtrics [5]. Participant responses were exported to .csv files and combined with their Prolific demographic data. Data was secured on the Qualtrics platform or on encrypted hard drives and all data handling ethics were approved by Lancaster University. Four duplicate entries, 87 incomplete submissions and one entry whose participant revoked consent after submission were removed. The Prolific identifier for each entry was then removed to anonymise the data.

2.3 Pre-screening

It is suggested participants from paid services, such as Prolific, often exaggerate their skills to be included in more studies [15, 30]. Therefore, to ensure high quality reliable responses, we pre-screened participants. Danilova et al., [15] recommend several questions to pre-screen participants, which we adapt to identify developers who use SAST tools. We first filtered participants using Prolific default options: computer programming skills, an approval rate of 95-100 (i.e., rate of completed studies: abandoning studies lowers approval rate) and an employment role in coding, technical writing or system administration. We created a short survey, separate from our main survey, with two questions (Q0.1 and Q0.2 found in Table 1). Q0.1 asks participants which vulnerability detection techniques they used; participants who did not select SAST tools were not eligible for the main survey. Q0.2 is a multiple choice question where participants were provided with a selection of real and fake vulnerability detection tools. Participants who claim to use fake tools were removed from the analysis. Overall, we gathered 1,263 responses from participants: 23% (286/1263) responded to Q0.1 as using SAST tools and were eligible for our study. However, 21% (260/1,263) failed our quality check by selecting fake tools and were removed from analysis, leaving 1,003 participants. Reid et al., [30] reported similar results with 33% of participants providing inconsistent responses and Danilova et al., [15] reported that 42% of their participants did not meet the criteria they claim to. Therefore, the 286 participants who selected SAST tools was reduced to 204. The remainder of the survey was distributed to the 204 participants, of whom 175 responded.

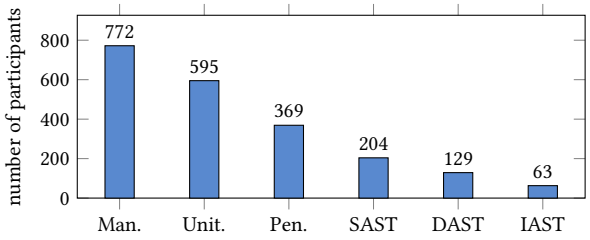

Figure 1: Q0.1: Techniques used to detect vulnerabilities.
Manual Code Review (Man.), Unit Testing (Unit.), Penetration Testing (Pen.), Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Interactive Application Security Testing (IAST).

2.4 Demographics

To further strengthen confidence in the generalisability of our study, we compared our participants' demographic information with previous results. The demographics of our participants closely matched the demographics from Stack Overflow's 2023 Developer Survey results [3] with some notable exceptions. First, the age of our participants was slightly lower, likely due to recruitment from a paid service attracting less financially stable participants. Second, our study contained more participants residing in countries close to the UTC timezone (United Kingdom, Portugal, South Africa) and fewer respondents residing in the USA, possibly due to the time our surveys were posted (0900 UTC), as positions were filled quickly.

2.5 Statistical Analysis

In our study, we obtain categorical data at nominal, ordinal and binary levels. We follow previous approaches and perform Chi-Square Tests of Independence to identify associations between two categories [25, 28]. We set the statistical significance level at 0.05 and present the results with the Chi-Square result (X-squared), Degrees of Freedom (df) and probability value (p-value).

3 Results

3.1 Pre-screening Results

Figure 1 suggests that SAST tools are used by 20% (204/1003) of developers. Previous work has concluded that ASAT use is widespread throughout OSS (between 56% and 66%) [10, 31]. However, we find SAST tool use to be much lower, possibly suggesting SAST tools are much less ubiquitous than ASATs. Previous work analysed repositories, whereas we survey developers, so it could be that 20% of developers contribute to 56%-66% of repositories. It may also be explained by different roles dealing with security violations, such as security specialists.

> **Finding 1:** SAST tools are popular vulnerability detection tools, but may be less widespread than generic ASATs; 20% (204/1003) of developers used SAST tools.

3.2 RA1: Current SAST Use

Table 2 suggests that most developers use SAST tools in their CI/CD pipelines (62%). Integrating static analysis tools into Continuous Integration (CI/CD) pipelines ensures code is being scanned regularly. This relatively high CI/CD SAST use may be related to our later finding that 42% of employers enforced SAST use and we assume this is done via CI/CD integration.

Table 2: Q2: How developers use SAST tools.

Response	Count	% (n=175)
CI/CD Pipeline	108	61.7%
Web Interface	82	46.9%
Command Line Interface	80	45.7%
IDE Plugin	69	39.4%
Standalone application	29	16.6%

Table 3: Q4: Why developers use SAST tools.

Response	Count	% (n=175)
They are good at detecting vulnerabilities	136	77.7%
They are convenient to use	99	56.6%
I am encouraged by my employer	86	49.1%
SAST tool usage is company policy	73	41.7%
I am encouraged by my colleagues	50	28.6%

Table 4: Q5: How effective SAST tools are.

Response	Count	% (n=175)
Not effective	0	0.0%
Slightly effective	10	5.7%
Moderately effective	92	52.6%
Very effective	66	37.7%
Extremely effective	7	4.0%

Table 3 presents responses as to why developers use SAST tools (Q4 Table 1) and suggests that most developers use SAST tools because they believe they are good at detecting vulnerabilities (78%) and convenient to use (57%). However, when asked how effective SAST tools are (Q5 Table 1), most developers (53%) believed SAST tools to be only moderately effective (Table 4), suggesting improvements can still be made to the detection rate of SAST tools to increase their effectiveness.

> **Finding 2:** Developers believe SAST tools to be good at detecting vulnerabilities and convenient to use; however, only moderately effective, suggesting improvements can be made to their detection rate to improve effectiveness.

Figure 2 shows the different SAST tools used by developers. SonarQube is the most popular tool, used by 59% of developers. Despite this, developers do use a variety of tools; in the 'Other' category, 30 developers named 26 unique tools. However, the majority of developers only use a single tool.

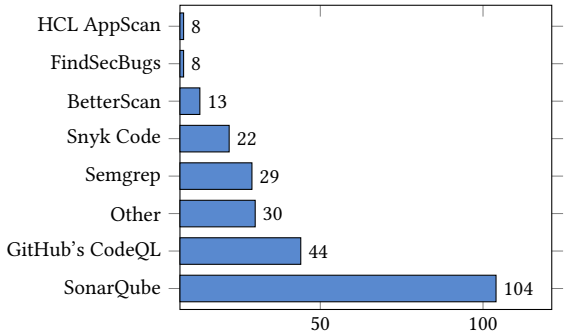

Figure 2: Q7: Top eight SAST tools developers use.

Table 5: The number of tools used and configuration status.

# of tools	Configure	Don't configure	Total
Single	32	65	97
Multiple	43	24	67
Total	75	89	164

Contingency Table 5 presents the number of developers who use a single or combine multiple SAST tools and whether they configure their tools. Previous work recommend configuring or combining multiple SAST tools as doing do may detect more vulnerabilities [11, 14, 18, 19, 27, 29]. However, many developers do not configure SAST tools (54%) or use more than a single tool (59%) and 40% do neither, potentially missing many vulnerabilities.

RA1 Summary: Most developers have integrated SAST tools into their CI/CD pipelines. However, most developers do not combine multiple tools or configure their tools, and many do neither.

3.3 RA2: Tool Configurability

The rule sets of SAST tools can be typically configured by the user, such as enabling, disabling, editing or even creating new rules. We asked developers for each SAST tool they use: what type of configurations they perform and how difficult each tool is to configure.

Table 6: Developers combining and configuring SAST tools.
Combine (Comb), Configure (Conf), Enable (Ena), Disable (Dis), Edit (Edi), and Create (Cre)

Tools	Total	Comb	Conf	Ena	Dis	Edi	Cre
SonarQube	99	38	41	33	26	24	16
CodeQL	37	28	23	17	10	14	13
Semgrep	28	20	17	16	12	10	5
Snyk Code	22	18	8	6	4	4	2
BetterScan	12	8	7	4	4	5	1
HCL AppScan	8	6	7	6	0	4	3
FindSecBugs	8	8	3	3	2	2	0
Veracode	3	3	2	2	2	2	0
Total	**247**	**150**	**124**	**100**	**69**	**77**	**44**

Table 6 presents the total number of developers that used each SAST tool and the type of configurations they performed. Note: figures are lower than Figure 2 because 11 participants provided no answer regarding configuring tools. Enabling standard rules is the most common type of configuration change performed on 41% of tools. These results suggest that more tools have rules enabled than disabled, and may suggest a discrepancy between the tool providers and their users in respect of what they want from the tool, i.e., balancing the true and false positives reported.

Table 6 also presents the rate of combination and configuration of each tool. Despite SonarQube being the most popular tool, it is one of the least configured tools, configured by 41% of its users. Developers not configuring their SAST tools may suggest there is some difficulty when configuring the tool or, alternatively, integrating SAST tools into CI/CD pipelines may make configuring tools less accessible. However, most tools were described as at least slightly easy to configure (74%), albeit by developers who already configure tools. In section 3.5, we report what metrics are associated with developers configuring and combining SAST tools.

RA2 Summary: The most common type of configuration was to enable rules which were previously disabled, whereas developers rarely created new rules.

3.4 RA3: Why do developers configure SAST tools?

Table 7: Q14: Why developers configure SAST tools.

	Ranking							
Statement	**1st**	**2nd**	**3rd**	**4th**	**5th**	**6th**	**Mean**	**St. dev.**
Detect more vulnerabilities	50	16	3	3	1	0	5.52	5.07
Add suggested fixes	6	12	14	14	24	3	3.95	3.60
Add additional information	3	16	19	20	15	0	3.62	3.29
Add rules created by someone else	6	19	23	15	10	0	3.37	3.13
Reduce number of warnings	7	9	13	21	21	2	3.36	3.14
Other	1	1	1	0	2	68	1.19	0.95

Table 7 presents developer rankings of why they configure SAST tools. Not surprisingly, the majority of developers who configured SAST tools prioritise detecting more vulnerabilities, with 88% of participants choosing this as their first or second most important reason. The large difference between 'detect more vulnerabilities' and 'reduce the number of warnings' could suggest that developers care more about catching missed vulnerabilities than decreasing the number of false positives.

RA3 Summary: Developers who configure SAST tools prioritize detecting more vulnerabilities over reducing the number of warnings considerably, which may suggest that developers prefer fewer false negatives over fewer false positives.

3.5 RA4: Metrics associated with configuring and combining SAST tools

Table 8: How developers use SAST tools x company size.

	1	2-9	10-49	50-249	250+	Total
Uses CI/CD	2	3	10	29	64	108
Does not use CI/CD	0	8	14	20	24	66
X-squared = 15.542, df = 4, p-value = **0.0037**						
Uses Standalone	2	4	7	4	12	29
Does not use Standalone	0	7	17	45	76	145
X-squared = 18.906, df = 4, p-value = **0.0008**						

Table 8 explores whether there is an association between how developers use SAST tools (e.g., command line, CI/CD pipeline) and whether they are based in small or large companies (in terms of number of employees). Table 8 suggests that larger companies favour integrating SAST tools into their build processes rather than using a standalone tool. It makes sense that companies move from using a dedicated SAST application to integrating SAST into build processes as the company grows. Integrating SAST tools into a CI/CD pipelines ensures that all code changes are scanned before

being pushed to a central repository and may even block commits depending on user specified security violations. Such integration also automates SAST scans and should make it easier for developers to use multiple tools.

Table 9: SAST tool deployment x combine SAST tools.

	Multiple tools	Single tool	Total
Uses CLI	45	35	80
Does not use CLI	26	68	94
X-squared = 13.466, df = 1, p-value = **0.0002**			
CI/CD	45	63	108
Does not uses CI/CD	26	40	66
X-squared = 0.018776, df = 1, p-value = 0.8910			

We found no significance difference between developers using SAST tools in their build processes and the rate at which they combine multiple tools. Table 9 presents how developers use SAST tools and whether they use multiple tools. We found significant results from developers using SAST tools through the Command Line Interface (CLI), which may be due to it being easier to automate compared to IDE plugins, web interfaces and standalone applications. However, the discrepancy between CLI and CI/CD results may suggest that developers have less control of their build process compared to the tools they use via a command line.

Finding 4: How developers interact with SAST tools seems related to company size (number of employees); larger companies seem to favour integrating SAST tools into their CI/CD pipelines. However, using SAST tools in CI/CD pipelines does not result in multiple tools being used.

Table 10: Experience using SAST tools x configure SAST tools.

	Configure	Do not configure
Less than a year	6	26
Between 1 and 5	57	49
Between 6 and 10	9	12
11+	3	2
Total	75	89
X-squared = 12.629, df = 3, p-value = **0.0055**		

With regard to developers configuring tools, we found only one metric to be statistically significant: the number of years of experience with SAST tools. Table 10 presents the results of our statistical analysis of developer years of experience using SAST tools and whether or not they configure them. These results suggest that new users do not configure SAST tools and it is likely that more effort is needed to assist new users to configure their tools.

RA4 Summary: Larger companies seem to prefer integrating SAST tools into CI/CD pipelines; however, this is not associated with more SAST tools being used. Using multiple SAST tools is associated with using tools through a command line. Whether developers configure tools seems related to prolonged experience with SAST tools.

4 Discussion

4.1 SAST tool use

Using SAST tools out of the box can provide some benefit and detect vulnerabilities missed by human error. However, combining tools will detect more vulnerabilities [14, 18, 19, 27, 29]. Li et al., [24] report that even the best performing SAST tool could only detect 13% of real Java vulnerabilities, but can be improved to 28% by combining tools.

Our results suggest that most developers do not configure (54%) or combine multiple SAST tools (59%) and many do neither (40%). The lack of SAST tool use potentially leaves many missed vulnerabilities and developers should consider either configuring a single tool or combining multiple tools to detect more vulnerabilities.

4.2 SAST deployment

On a positive note, 62% of developers integrate SAST tools into their CI/CD pipelines, allowing SAST tools to automatically scan code to help prevent vulnerabilities being introduced. Do et al., [17] recommend using multiple tools with a single reporting system, which CI/CD pipelines can help with. However, we found no significant association between developers combining tools and integrating them into their build process. Further work is needed to help developers combine multiple tools into their CI/CD pipeline, such as a standardised taxonomy of rules to help reduce the number of duplicate warnings from multiple tools.

Our results also suggest there is no association between the type of SAST deployment and whether developers configure their SAST tools. Zampetti et al., [34] reviewed configurations of ASATs in CI/CD pipelines and found that all tools were configured in a pipeline. Our results are much lower, in that many developers do not configure tools even in the CI/CD pipeline. Integrating SAST tools in CI/CD pipelines is favoured by larger companies. However, it is important to establish SAST tool use early in a project - less experienced users are unlikely to configure tools. More work to assist developers configure SAST tools such as training, tutorials or automating the process of turning missed vulnerabilities into new rules for SAST tools is clearly needed.

4.3 SAST tool configurations

Developers that configure SAST tools enable rules more than other types of configurations. Enabling rules was performed on 81% of configured tools, more so than disabling rules, which was performed on 56%. Our results suggest a discrepancy between SAST tool providers and their users.

Christakis & Bird [13] report that developers are often frustrated by ASATs due to the wrong checks turned on by default from their tool providers and conclude that program analysis should not have all rules on by default because a high false positive rate leads to disuse. However, Ami et al., [6] interviewed developers which expressed a different viewpoint: "The developer is the enemy" in that they were typically against developers regulating which warnings to ignore or disable. SAST tool providers may disable rules because they generate more false positives than true positives. However, SAST users enable them regardless. The majority of developers

configure tools to detect more vulnerabilities (67%), which suggests they prioritise reducing false negatives; a finding shared with Ami et al., [6], who report nearly all interviewed developers preferred fewer false negatives over fewer false positives. Our recommendation is that SAST tool providers should reconsider the priorities of their tools and enable all rules by default. SAST tool users should configure their tool for their needs, but be cautious when determining which warnings to ignore or disable.

4.4 SAST vs. ASATs

We discuss similar previous work in more detail in Section 5, but our study contradicts findings from previous work. Previous work suggests ASATs are widespread throughout OSS and developers rarely configure tools. We find specialised SAST tools to be less widespread than ASATs; only 20% (204/1,003) of participants used SAST tools. However, those who used SAST tools configured them at a higher rate than ASATs. Our results showed that 46% of developers configured their SAST tool.

It is well known that static analysis tools produce a large number of false positives; studies have concluded the high false positives rate is a burden to developers [20, 21] and recommend reducing the false positive rate of ASATs. However, we found that developers configured tools to detect more vulnerabilities, not to reduce the number of warnings and enable rules more so than disable them. Our results also suggest that developers prioritise reducing false negatives rather than reducing false positives, a finding shared by Ami et al., [6]. Our results highlight differences between ASATs and SAST tools; studies drawing conclusions about one type of static analysis tool should not generalise to all types.

4.5 Threats to validity

Any empirically-based study needs to consider the different validity threats (internal, external and construct) that may undermine its results.

Internal validity: In our study, we perform Chi-Square Tests of Independence on developer responses and discuss significant results. The results may be due to extraneous variables not discussed.

External Validity: We recruited participants from a paid service which is known to show inconsistent results [15, 30]. However, we mitigate this threat by pre-screening all participants for fraudulent responses.

Construct validity: Our survey may be vulnerable to selection bias. However, we compared the demographics of our participants with the wider population to ensure they were representative. To avoid response bias, we also avoided leading questions where possible.

5 Related Work

It is important to evaluate static analysis tools to establish a baseline of their performance so that they can be improved. There have been many studies that evaluate static analysis tools [8, 9, 12, 16, 19, 24, 26, 27]. Previous studies typically run a static analysis tool on a set of defects and report their performance on various metrics, such as precision, recall, time or usability. Li et al., [24] evaluated seven SAST tools on a set of real vulnerability data and reported that only 13% of vulnerabilities could be detected, showing poor recall in production settings; they suggest using a combination of tools. Our study evaluates a number of SAST tools by surveying developers to report a metric yet to be explored - their configurability. We report which tools are configured by developers and what type of configurations they perform.

Investigating human factors associated with using tools is also important to better integrate tools into developer workflows and to understand the reasons why tools may or may not be adopted [33]. Beller et al., [10] mined 122 repositories and surveyed 36 developers and reported that a majority of developers used a single ASAT tool; moreover, configurations very rarely changed. Vassallo et al., [31] investigated ASAT use in different contexts by surveying 56 developers and interviewing 11 industry experts. The study confirmed previous findings that ASATs were rarely configured; most developers (56%) only configured ASATs at the start of a project and 16% never configured them. Our work resembles previous work as we survey developers regarding their static analysis use. However, we focused on SAST tools rather than ASATs.

Ami et al., [6] performed in-depth interviews with 20 practitioners to uncover an industry perspective on SAST tools. Our study most resembles Ami et al.'s study but obtains quantitative data to complement their qualitative approach. Ami et al., report many findings, including that developers are more tolerant of false positives than the academic literature suggests and show a preference to fewer false negatives [6]. However, they do not question developers about their configuration of SAST tools. Rather, they ask about how developers address issues with their SAST tools, such as missed vulnerabilities. While being concerned with security, few developers cared about missed vulnerabilities, assuming other techniques would detect them. We share many of the findings by Ami et al.,; we found that the majority of developers used SAST tools as is, i.e., never addressing missed vulnerabilities. We also found that the primary motivator behind developers configuring SAST tools was to reduce the number of false negatives.

6 Conclusions and future work

In this emerging results paper, we surveyed 1,263 developers to understand how they used and configured SAST tools. We found that only 20% (204/1,003) of developers used SAST tools, but those who did use them typically used them in their CI/CD pipelines (62%). However, a large amount only use a single tool with its standard configuration (40%), potentially leaving many missed vulnerabilities. To aid in this sense, SAST tool providers should make their tools easier to configure and employers/managers should introduce SAST tools early in development and encourage the configuration of SAST tools. We highlighted differences between ASATs and SAST tools and suggest any future work which analyses static analysis tools to take care when generalising results to all types of static analysis tools. We also found that developers tended to enable rules more than disable them and prioritise detecting more vulnerabilities than reduce the number of warnings. Our results suggest a discrepancy between SAST tool providers and their users. We recommend first, that all rules be enabled by default by SAST tool providers and, second, caution developers to ensure they configure their tools according to their needs.

References

[1] 2017. CVE-2017-0144. https://nvd.nist.gov/vuln/detail/CVE-2017-0144. Accessed: 2024-06-06.

[2] 2021. CVE-2021-44228. https://nvd.nist.gov/vuln/detail/CVE-2021-44228. Accessed: 2024-06-06.

[3] 2023. Developer Survey. https://survey.stackoverflow.co/2023/. Accessed: 2024-06-04.

[4] 2024. Prolific. https://www.prolific.com/. Accessed: 2024-06-04.

[5] 2024. Qualtrics XM. https://www.qualtrics.com. Accessed: 2024-06-06.

[6] Amit Seal Ami, Kevin Moran, Denys Poshyvanyk, and Adwait Nadkarni. 2023. " False negative–that one is going to kill you": Understanding Industry Perspectives of Static Analysis based Security Testing. *arXiv preprint arXiv:2307.16325* (2023).

[7] Anon. 2024. Do Developers Use Static Application Security Testing (SAST) Tools Straight Out of the Box? A large-scale Empirical Study. https://zenodo.org/records/11488719. Accessed: 2024-06-05.

[8] Aman Anupam, Prathika Gonchigar, Shashank Sharma, Prapulla SB, and Anala MR. 2020. Analysis of Open Source Node. js Vulnerability Scanners. *International Research Journal of Engineering and Technology (IRJET) e-ISSN* (2020), 2395–0056.

[9] Andrei Arusoaie, Stefan Ciobâca, Vlad Craciun, Dragos Gavrilut, and Dorel Lucanu. 2017. A comparison of open-source static analysis tools for vulnerability detection in c/c++ code. In *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 161–168.

[10] Moritz Beller, Radjino Bholanath, Shane McIntosh, and Andy Zaidman. 2016. Analyzing the state of static analysis: A large-scale evaluation in open source software. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 470–481.

[11] Gareth Bennett, Tracy Hall, Emily Winter, and Steve Counsell. 2024. Semgrep*: Improving the Limited Performance of Static Application Security Testing (SAST) Tools. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering* (Salerno, Italy) *(EASE '24)*. Association for Computing Machinery, New York, NY, USA, 614–623. https://doi.org/10.1145/3661167.3661262

[12] Paul E Black, Damien Cupif, Guillaume Haben, Alex-Kevin Loembe, Vadim Okun, and Yann Prono. 2023. SATE VI Report. (2023).

[13] Maria Christakis and Christian Bird. 2016. What developers want and need from program analysis: an empirical study. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*. 332–343.

[14] Roland Croft, Dominic Newlands, Ziyu Chen, and M Ali Babar. 2021. An empirical study of rule-based and learning-based approaches for static application security testing. In *Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*. 1–12.

[15] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. 2021. Do you really code? Designing and Evaluating Screening Questions for Online Surveys with Programmers. *CoRR* abs/2103.04429 (2021). arXiv:2103.04429 https://arxiv.org/abs/2103.04429

[16] Aurelien Delaitre, Bertrand Stivalet, Paul Black, Vadim Okun, Terry Cohen, and Athos Ribeiro. 2018. SATE V Report: Ten Years of Static Analysis Tool Expositions. (2018-10-23 2018). https://doi.org/10.6028/NIST.SP.500-326

[17] Lisa Nguyen Quang Do, James R Wright, and Karim Ali. 2020. Why do software developers use static analysis tools? a user-centered study of developer needs and motivations. *IEEE Transactions on Software Engineering* 48, 3 (2020), 835–847.

[18] Sarah Elder, Nusrat Zahan, Rui Shu, Monica Metro, Valeri Kozarev, Tim Menzies, and Laurie Williams. 2022. Do I really need all this work to find vulnerabilities? An empirical case study comparing vulnerability detection techniques on a Java application. *Empirical Software Engineering* 27, 6 (2022), 154.

[19] Christoph Gentsch. 2020. Evaluation of open source static analysis security testing (SAST) tools for c. (2020).

[20] Nasif Imtiaz, Akond Rahman, Effat Farhana, and Laurie Williams. 2019. Challenges with responding to static analysis tool alerts. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 245–249.

[21] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 672–681.

[22] Mark Kasunic. 2005. Designing an effective survey.

[23] Harjot Kaur, Sabrina Amft, Daniel Votipka, Yasemin Acar, and Sascha Fahl. 2022. Where to recruit for security development studies: Comparing six software developer samples. In *31st USENIX Security Symposium (USENIX Security 22)*. 4041–4058.

[24] Kaixuan Li, Sen Chen, Lingling Fan, Ruitao Feng, Han Liu, Chengwei Liu, Yang Liu, and Yixiang Chen. 2023. Comparison and Evaluation on Static Application Security Testing (SAST) Tools for Java. (2023).

[25] Johan Linaker, Sardar Muhammad Sulaman, Martin Höst, and Rafael Maiani de Mello. 2015. Guidelines for conducting surveys in software engineering v. 1.1. *Lund University* 50 (2015).

[26] Rahma Mahmood and Qusay H Mahmoud. 2018. Evaluation of static analysis tools for finding vulnerabilities in Java and C/C++ source code. *arXiv preprint arXiv:1805.09040* (2018).

[27] Francesc Mateo Tudela, Juan-Ramon Bermejo Higuera, Javier Bermejo Higuera, Juan-Antonio Sicilia Montalvo, and Michael I Argyros. 2020. On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications. *Applied Sciences* 10, 24 (2020), 9119.

[28] Mary L McHugh. 2013. The chi-square test of independence. *Biochemia medica* 23, 2 (2013), 143–149.

[29] Paulo Nunes, Ibéria Medeiros, José Fonseca, Nuno Neves, Miguel Correia, and Marco Vieira. 2019. An empirical study on combining diverse static analysis tools for web security vulnerabilities based on development scenarios. *Computing* 101 (2019), 161–185.

[30] Brittany Reid, Markus Wagner, Marcelo d'Amorim, and Christoph Treude. 2022. Software Engineering User Study Recruitment on Prolific: An Experience Report. *CoRR* abs/2201.05348 (2022). arXiv:2201.05348 https://arxiv.org/abs/2201.05348

[31] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Harald C Gall, and Andy Zaidman. 2020. How developers engage with static analysis tools in different contexts. *Empirical Software Engineering* 25 (2020), 1419–1457.

[32] Stefan Wagner, Daniel Mendez, Michael Felderer, Daniel Graziotin, and Marcos Kalinowski. 2020. Challenges in survey research. *Contemporary Empirical Methods in Software Engineering* (2020), 93–125.

[33] Emily Winter, Vesna Nowack, David Bowes, Steve Counsell, Tracy Hall, Sæmundur Haraldsson, and John Woodward. 2022. Let's talk with developers, not about developers: A review of automatic program repair research. *IEEE Transactions on Software Engineering* 49, 1 (2022), 419–436.

[34] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. 2017. How open source projects use static code analysis tools in continuous integration pipelines. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 334–344.