Context Matters: Qualitative Insights into Developers' Approaches and Challenges with Software Composition Analysis

Elizabeth Lin, Sparsha Gowda, William Enck, and Dominik Wermke
North Carolina State University
{etlin, ssgowda3, whenck, dwermke}@ncsu.edu

Abstract

Software Composition Analysis (SCA) is an important part in the software security lifecycle. Establishing the individual software components and versions that make up an application allows for identifying and remediating vulnerabilities. However, SCA tools have not kept up with the ever growing number of new vulnerabilities each year. Developers are flooded with vulnerability alerts and often struggle to quickly remediate critical issues with external components.

We conducted 20 interviews with developers to investigate their processes and challenges around using SCA in their software projects. Interviews covered how SCA tools are integrated into workflows, how reports are interpreted and acted upon, and what challenges were encountered. We find that SCA tools are most often integrated into build pipelines and that users report that information in SCA alerts is too generic and lack context, specifically context on infrastructure, network configurations, reachability, and exploitability. Based on our findings we conclude that context matters throughout the SCA process, including for evaluating impact, when to trigger SCA scan runners, and how to integrate and communicate tool findings.

1 Introduction

Software Composition Analysis (SCA) is a method for identifying and analyzing software components. SCA tools identify third-party components and libraries in software applications, enabling the tracking and mitigation of vulnerabilities in these components and the management of software licenses. External software components make managing vulnerabilities challenging, as they are often outside the direct control of the developers. Various studies estimate more than 75% of applications include third party open source software [1], [29], that 96% of codebases contain open source software, and that 77% of the code in the codebases originate from open source [1]. Consequently, software composition analysis is becoming increasingly important, allowing the industry to identify and manage vulnerabilities in their third-party components.

The need for SCA tools is further underscored by the rise of published vulnerability reports. In 2019, 17,308 CVEs were published. Five years later in 2024, the number of published CVEs has grown to 29,004 [12]. This growing number of reported vulnerabilities has caused companies to integrate SCA tools into their development pipelines to filter and manage security risks. The "Log4Shell" arbitrary code execution vulnerability in the Log4j Java logging framework is a widely referenced high-impact vulnerability in a third-party component [26]. Reported in 2021, companies struggled to identify whether and where Log4j was used in their systems and even today there remain vulnerable systems in the wild [47]. The US government has again emphasized the importance of managing vulnerabilities in third-party components in the 2025 Executive Order 14144 [16]. SCA tools help following federal requirements such as these by identifying third-party risks and generating compliance artifacts such as SBOMs.

While SCA tools simplify managing security risk for third-party components, the large number of alerts they output can be a challenge. Two factors contribute to the large number of alerts: (1) the large number of published CVEs and (2) a large dependency tree from all the imported external components. The large number of alerts can result in alert fatigue [44], where developers ignore vulnerability alerts because of being overwhelmed by them. Managing vulnerabilities in external libraries is challenging, as they are often maintained by others and may include additional dependencies. Developers also must often wait for maintainers to address issues in their external components, further complicating resolution [20].

Previous studies on SCA have looked into how different SCA tools compare against each other [21], [41], [55]. However, they lack understanding on how SCA users interact with the tools and the actions they take after receiving results from the tool. Often, software vulnerabilities in external libraries are resolved when developers update to non-vulnerable release. Updating external libraries may also introduce compatibility issues in applications. For businesses that rely on

software systems for their everyday operations, an unexpected issue can interrupt or even halt their business.

To address the gaps around integration and encountered challenges from previous studies, we conducted an interview-based qualitative study with 20 professionals focused on SCA users' experiences based on 4 research questions:

RQ1: *How do users interact with SCA tools?* We want to understand how users interact with SCA tools. There are two parts to this question: (1) how the SCA tools are run and (2) actions after receiving output from an SCA scan. We want to investigate if and how SCA tool integrate into the software development pipeline and how frequently SCA scans are run. We also want to understand how the SCA tool outputs are interpreted and the actions taken based on these outputs.

RQ2: *What are the challenges when deploying SCA tools?* Our second research question is an extension to the first part of RQ1. We want to identify challenges related to running the tools, as well as investigate if new challenges emerge if organizations scale the SCA process to more or larger projects.

RQ3: *What are the challenges when acting on SCA results?* Our third research question is an extension to the second part of RQ1. We want to understand if there are challenges related to resolving the alerts. Resolving issues in external code may be more complicated as the user may not have a full understanding of imported code.

RQ4: *How can the SCA process be improved?* From the challenges and issues we identified in previous RQs, we want to identify improvements that could be made to SCA processes and tools.

By addressing these questions and identifying areas for improvement, we aim to support the further adoption of SCA tools and drive their development and deployment towards more effective and accessible vulnerability management.

Key Findings: We observe that additional context is needed across the entire SCA process: (1) The lack of vulnerability context requires security engineers to manually review the risk and impact of the vulnerability; (2) Vulnerabilities in external components are continually discovered and SCA scans can halt build pipelines for unrelated reasons, therefore when and how the SCA scan runs in the development cycle matter; and (3) Scaling the SCA process to multiple teams and projects can generate overhead when communicating results and fixing vulnerabilities. Context matters throughout the entire SCA process, and more and better context relating to infrastructure, network configurations, reachability, and exploitability will help streamline vulnerability management. The integration of SCA should take into account context of the application, the vulnerability, and also the organization's development pipelines.

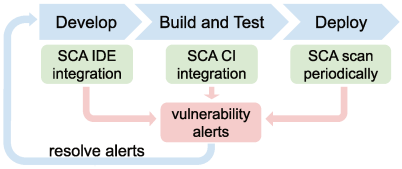

Figure 1: Common ways SCA tools are integrated into software lifecycles.

2 Background and Related Work

In this section we discuss topics relevant to our study and prior research in those areas.

Software Composition Analysis: SCA identifies software components in an application. Initially, SCA was used for open source license management, and Ombredanne [32] discusses challenges in using SCA for license compliance. However, as security concerns with the software supply chain grew, SCA expanded to vulnerability management. Note that SCA should not be confused with Static Application Security Testing (SAST) tools, which are used to identify new vulnerabilities within the main application. SCA complements SAST, providing insight into known vulnerabilities in imported software components. Figure 1 shows three common ways SCA is integrated into software lifecycles.

Few studies have performed in-depth treatments of SCA tools and security. Imtiaz, Thorn, and Williams [21] compared the analysis reports of 9 industry-leading SCA tools on OpenMRS, a large web application. They found that different tools report different results and proposed future directions related to false positives and continuous monitoring. A more recent study [41] had similar findings. Several studies [13], [14], [31], [55] have investigated the Java ecosystem, identifying the challenges it presents for SCA tools, which include those that result from code modifications [13] and cloning or shading code from software components [14].

While this prior work has considered how SCA tool results differ, it fails to understand how developers interpret the results for actions such as vulnerability remediation. We want to understand if there are any prioritization processes for the results, as the large number of security vulnerabilities can overwhelm developers [19], [42]. Then we want to understand how the results are received by users, how they determine if the vulnerability requires their attention and needs to be resolved. Finally, we want to understand what parts of determining and resolving the vulnerability is automated.

Vulnerability Data and Management: Software vulnerability data is required for SCA tools to identify vulnerabilities in applications. The National Vulnerability Database (NVD) is the oldest and most widely used vulnerability database. NVD uses the Common Vulnerabilities and Exposures (CVE) system, which includes a CVE ID, a description of the vulnerability, and public references. NVD also often enriches CVEs

with a CVSS severity score [40], though scaling issues has recently raised doubts into the viability of doing so in the long term [30], [43], [53]. Each vulnerability database has its own identifier prefix, though they unilaterally reference the CVE ID as a canonical identifier. Vulnerabilities are colloquially known as "CVEs."

Prior work has extensively considered a range of vulnerability data and management topics [24], [28], [34], [38], in various ecosystems [2], [25], [49], [54], [57]. Vuln4Real [35] measures vulnerable dependencies in the Java ecosystem. Wu *et al.* [52] investigated the impact of a vulnerability in a dependency on an application based on call graph paths and reachability of functions. Pashchenko, Vu, and Massacci [36] conducted a qualitative study with developers to understand the factors that impact dependency selection decisions. Plate, Ponta, and Sabetta [37] proposed an approach to assess the impact of code changes in security fixes.

Unfortunately, the lack and inconsistency of vulnerability data [4] makes it hard to determine the impact of a vulnerable dependency on an application, including if it matters at all. Prior studies [9], [15], [39] have pointed out the need to automatically discover software patch data for vulnerability discovery and repair.

Interview Studies: Qualitative research uses surveys or interviews to gather user opinions and is a growing research area in the security literature. Multiple qualitative studies have emerged to address challenges with vulnerability management [3], [5], [6], [22], [48]. Barrett *et al.* [8] conducted 12 interviews with sysadmins, managers, team leads, and others in various roles about the security issues, concerns, and challenges in their work. Gutfleisch *et al.* [18] interviewed developers about usability of their secure software development processes, identifying a high impact of contextual factors such as stakeholder pressure, presence of expertise, and collaboration culture. Johnson *et al.* [22] interviewed developers on their use of static analysis tools and desired improvements.

Prior qualitative studies have also considered the open source ecosystem. Several studies [33], [45], [51] have discussed trust and barriers in the open source ecosystem. Wermke *et al.* [50] discussed broad security-related challenges in open source. The open source community relies on developers who contribute voluntarily, and there could be challenges relating to issues such as code quality and maintenance [46]. Zhou, Vasilescu, and Kästner [56] discussed 'forks' in open source and Bogart *et al.* [10] discussed how changes in open source can break software.

3 Methodology

Over a 6 month period starting October 2024, we conducted 20 semi-structured interviews with industry professionals with SCA experience. The interviews included discussions on how the SCA tools were used, encountered challenges, and how they can be improved. In this section, we outline our interview guide, recruiting participants, conducting interviews, and how we analyzed the interview data.

3.1 Study Setup

We opted for semi-structured interviews for our study to gather deep insights from participants. Semi-structured interviews allow us to (1) ask questions we prepared and (2) dive deeper into any interesting insights the participants have.

We based our interview guide on our research questions and cognitive walk-throughs of six popular SCA tools. We further refined our interview guide through feedback rounds with other researchers and a pilot interview. Our final interview structure covers questions related to the process of using and integrating SCA tools, how SCA reports are interpreted, as well as challenges when using the tools.

Cognitive Walk-Throughs: We wanted to understand the full process related to SCA tools, from the selection of SCA tools to resolving vulnerability alerts outputted from the SCA tool. Therefore, before conducting the interviews, two researchers conducted cognitive walk-throughs of six popular SCA tools: (1) Snyk Open Source,[1] (2) Grype,[2] (3) GitHub Dependabot,[3] (4) Endor Labs SCA,[4] (5) OSV-Scanner,[5] and (6) Semgrep Supply Chain.[6] We opted for both open source and proprietary tools to cover a wide range of tools. The six SCA tools were selected based on the following criteria: (1) the tool had to be freely available to us and (2) the tool had to generate a report of the vulnerabilities existing in the software components because one of our research questions is concerned with the handling of such reports. The selected tools only assisted us in the walk-throughs and are not necessarily the tools our participants use.

Our cognitive walk-through was focused on completing a main task: *running the the SCA tool on a sample application and generating a vulnerability report* and a subtask: *rank or filter the vulnerability alerts from the tool*. We focused on the following guiding questions while completing the task to identify additional challenges or issues our interview guide could focus on: (1) How is the tool used? (2) What do we need to provide to the tool? (3) What does the tool output? (4) What information is provided by the SCA tool? (5) What (additional) features are provided by the SCA tool? These walk-throughs served as one of the bases of questions we asked in our interviews.

Interview Guide: We based our interview guide on our 4 main research questions. The structure of the interview guide and general questions were initially informed by our research

[1] https://snyk.io/product/open-source-security-management/
[2] https://github.com/anchore/grype
[3] https://docs.github.com/en/code-security/dependabot
[4] https://www.endorlabs.com/use-cases/reachability-based-sca
[5] https://github.com/google/osv-scanner
[6] https://semgrep.dev/products/semgrep-supply-chain/

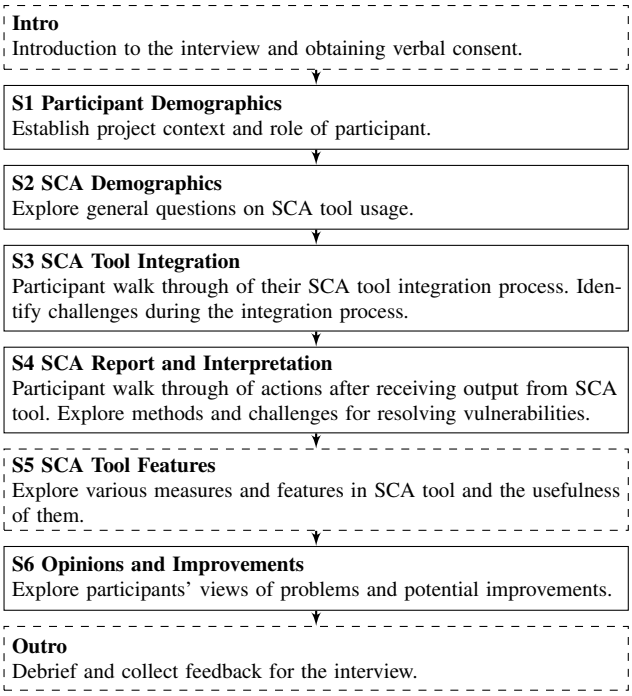

Figure 2: Illustration of the flow of topics in the semi-structured interviews. In each section, participants were presented with general questions and corresponding follow-ups, but were generally free to diverge from this flow at will. Solid boxes were required and dotted boxes were flexible sections.

questions, while the cognitive walk-throughs of SCA tools provided us with specific questions and follow-ups. We also collected feedback from researchers with interview experience on our interview guide. We conducted one pilot interview and made minor updates to the interview guide based on the feedback. As there were no major changes needed after the pilot and feedback, we decided to proceed with interviewing actual participants. We conducted the initial interviews provisionally to test our setup. As no major changes to the interview guide were needed, we decided to included these interviews in the final dataset.

After each interview, we reviewed our interview guide and assessed any new information we learned. After six interviews, we identified two new topics related to challenges with SCA tools. We added a follow-up question on whether SCA tools perform differently for different programming languages and ecosystems. We added another follow-up question on compatibility issues when fixing alerts through updating dependency versions. Furthermore, we found that discussion around tool features (S5) would often be discussed with tool improvements, thus we moved the topic to a lower priority and only asked it if we had extra time in the interviews. The interview guide remained unchanged after interview 12. We provide the final interview guide in Appendix A.

Interview Structure: Our interviews were based around non-leading, open questions with specific sub-questions as follow-ups, allowing interviewees to elaborate their thoughts and answers. Figure 2 shows our final interview structure. We started the interviews with introduction questions about the participant's organization and role (S1). Next, we proceeded to ask about their SCA experience (S2), where we tried to understand general information like which tool is used and how the experience with the tool was. We then proceeded to ask questions about how they integrate SCA tools into their pipelines (S3). In this section, we also followed up on challenges when integrating the tools. After understanding how SCA tools are integrated, we continued and asked questions on how the SCA reports are interpreted and managed for resolving vulnerabilities identified from the SCA tools (S4). We also focused on any challenges the users run into when interpreting the SCA results. Finally, we asked users about their opinions on SCA tools and how they believe the tools could be improved (S6).

3.2 Participant Recruitment

We sought to recruit participants from the population of developers that have used (or even developed) SCA tools in their projects in the past. Because this is a highly specialized population and we wanted to gather insights about SCA usage in a wide range of projects and companies, we used a range of recruitment strategies to reach a wide and diverse pool of participants. Specifically, we used the following strategies:

Online Communities: Various online open source and security communities provide a forum for developers to engage in discussion. These communities are often in Discord servers and Slack channels. We searched the web for such communities with 'security', 'sec', or 'open source' in their names. We selected 10 online communities with appropriate channels for us to recruit participants and invited members to participate in our interview study.

Freelance Platforms: Freelance platforms include a wide variety of developers, which can broaden our participant population. We recruited developers with SCA experience through the Upwork[7] platform. We posted our study details on the platform and sent individual invites to freelancers that were a good fit for our study in terms of experience with SCA tools.

Conference Participants: We also attended local developer waterholes like a cybersecurity conference, InfoSeCon, and a security community event, Software Supply Chain Community Day, organized by our university. We networked and invited interested developers with SCA experience to participate in our interview study.

Snowball Sampling: We also asked interview participants if they knew of anyone else that would be a good fit for our

[7] https://www.upwork.com/

Table 1: Participant Demographics

P No.	Role†	Company Type	Years of Experience	Country	Codes	Duration
P01	Software engineer	Consulting firm	4	US	37	0:48:50
P02	Security engineer	Fintech company	9	Germany	33	0:45:55
P03	Security engineer	Survey company	14	US	47	0:52:11
P04*	Software engineer	Software vendor	21	US	34	0:48:13
P05	Security engineer	Telecommmunications	5	Norway	31	0:45:01
P06	Security architect	Healthcare	6	US	31	0:41:41
P07	Lead engineer	Travel arrangements	15	US	29	0:44:57
P08*	Director	Consulting firm	15	The Netherlands	28	3:06:25‡
P09	Security architect	Software vendor	35	US	48	1:00:16
P10	Software engineer	Software vendor	11	US	33	0:55:15
P11	Director	Electronics provider	30	US	42	0:56:25
P12*	Freelancer	Consulting	20	The Netherlands	22	0:49:46
P13	Security engineer	Career platform	1	India	23	0:49:28
P14	Security manager	IT solutions	5	Pakistan	35	0:53:02
P15	Security lead engineer	Non-profit organization	25	Norway	25	0:44:38
P16*	Security lead engineer	Software vendor	7	US	31	0:52:04
P17	Security engineer	Healthcare	3	US	28	0:35:58
P18	Security lead engineer	Healthcare	15	US	30	0:41:51
P19	Security engineer	Healthcare	11	US	29	0:37:16
P20	Security engineer	Software vendor	4	US	21	0:34:52

*Participant has experience with developing SCA tool
†Participant roles binned to preserve privacy
‡Participant voluntarily exceeded planned interview time

study. If they recommended someone, we would reach out and invite them to participate as well.

Of all the strategies, posting in online communities turned out to be the most effective recruitment method. We recruited 9 participants through online posts. Recruitment through freelance platforms, conferences, and community events also turned out to be a good strategy. 6 participants were recruited from the freelance platform, 4 of our participants were recruited through connecting at conferences, and 1 was from snowball sampling.

Selection criteria: To be eligible for our interviews, the participants had to be at least 18 years of age and certify that they have had experience with SCA tools. Before the interview, we asked the participants to answer the following questions: (1) What SCA tools have you used? (2) Could you briefly describe your experience with SCA tools? The questions allowed us to verify the participants have used SCA tools and would be able to answer our interview questions. All participants we interviewed had used SCA tools in the past and provided sufficient insights so that we could base our analysis on all 20 interviews.

Participants: Table 1 shows the demographic information of our participants. Our participants come from a wide range of industries, including but not limited to software vendors and consulting firms. The participants also have experiences that cover a broad spectrum, from one to thirty-five years of experience. The majority of our participants are based in the US, but there are also participants from Europe and Asia. As compensation of sharing their experiences and valuable time for 45 to 60 minutes, we offered each participant $60 or the equivalent value in Amazon.com vouchers.

3.3 Interview Procedure

Our interview study follows the ethical principles outlined in the Menlo report [7] and was approved by our university's IRB, which we discuss more in *Ethics Considerations* at the end of this paper. Prior to the interview, we sent the consent form to the participant to inform them of their rights, the goal of our study, and how they would benefit. Our interviews were conducted and recorded over zoom. The interviews generally lasted around 45 to 60 minutes; see Table 1 for per-interview duration. At the start of the interview, we reiterated the goal of our research and gave participants the chance to ask questions before we started the interview and recording. The recordings were later processed and analyzed, which we discuss in the next section (Section 3.4). After the interview, we again gave participants the opportunity to ask any questions and encouraged them to reach out with any questions or concerns.

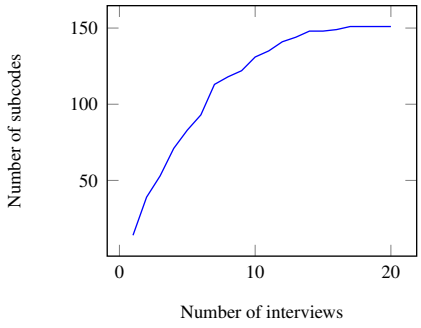

Figure 3: Count of subcodes in the codebook over number of interviews. A flattening out curve in later stages indicates reaching thematic saturation.

3.4 Coding and Analysis

To analyze our interviews, we applied a thematic analysis approach [11] using hybrid (combination of inductive and deductive) qualitative coding with two coders, a main coder and an assisting coder, who both also performed the cognitive walk-throughs mentioned in Section 3.3. The 20 interview audio files averaged to 54 minutes 18 seconds for each interview. The audio was transcribed into text transcripts using a locally-run *OpenAI whisper* machine learning model to ensure privacy of all participant recordings. Before starting coding, the lead researcher checked and corrected all transcripts for transcription mistakes.

We adopted a hybrid coding approach for our codebook development. First, we formulated an initial codebook structure based on our interview guide, with codebook sections corresponding to sections in our interview guide. Based on the first few interviews, our main coder did a first pass and added new subcodes to the codebook. During coding, we modified the initial codebook based on new themes and additional topics that emerged during subsequent interviews. Both coders participated in the cognitive walkthrough to ensure they have sufficient understanding of SCA tools. Before coding the transcripts, the two coders met to discuss the codebook to ensure they have the same understanding of the codes. Both coders iteratively and independently coded all interview transcripts. After each coded transcript, coders exchanged coding results.

The main coder led conflict resolution, addressing straightforward differences (e.g., including an additional word) at their discretion and adding second coder's codes they agreed with. Conflicts were minor and included missing subcodes, thus they were easily resolved. This iterative process continued until all conflicts were resolved. As all conflicts were resolved, we do not report intercoder agreement (IRR) [27]. Prior work has also adopted conflict resolution with similar approaches [23], [51].

Figure 3 shows the total number of subcodes in our codebook after each coding update. The total number of subcodes increases quickly in our first few interviews and levels off in later interviews, indicating that we reached saturation [17]. In total we assigned 637 codes resulting an average of 32 codes per interview. We provide the final codebook in Appendix B and discuss identified themes in Section 5.

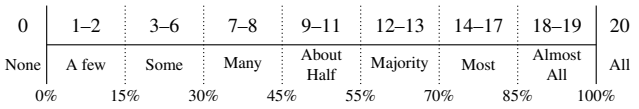

Figure 4: Interview reporting ranges to help with readability of results.

3.5 Limitations

Our work includes a number of limitations typical for this type of interview study and should be interpreted in context. Generally, self-report studies may suffer from biases such as: over- and under- reporting, sample bias, and social-desirability bias. We tried to minimize these biases through removing potential sources of bias in the interview procedure and by asking non-leading questions. Our work is a convenience sample and may not fully represent the entire population of SCA users, our study aimed to reach a broad and diverse sample through a number of sampling channels. We conducted the interviews in English, thus we cannot provide insight into SCA practices in (entirely) non-English speaking regions.

4 Results

This section discusses the results from our interviews. We divide our findings into the following subsections and discuss the related observations: (1) SCA tool demographics, (2) tool integration, (3) tool report, (4) developer opinions, and (5) improvements. The sections are largely based on the structure of our interview guide (Figure 2). We replace the tool name with a tool ID when a tool is mentioned in a quote to preserve confidentiality of participants, we also indicate open source tools by adding *(OsT)* after the tool ID in the quotes. We included some participant quotes that relate to SCA development experience as they provide additional insight into SCA tools. To distinguish between between user and development experience, quotes stemming from SCA development experience are labeled as *participantID-dev*. When reporting the percentage of participants who expressed a related opinion, we use specific phrases (e.g., "a few," "many," "most") for better flow and readability. The ranges we used for each of the terms is shown in Figure 4.

4.1 SCA Demographics

The first main topic discussed in our interviews was the use of SCA tools, including which SCA tool is used, reasons for selecting the specific tool, and other related SCA use. We

Table 2: Tool Demographics

Type	Times Mentioned	Tool ID
Fully open source, no vendor support	4	T03
	2	T09
	1	T21, T25, T26, T27
Open source, with vendor support	2	T17
	1	T23, T24, T29, T30
Open core, with vendor support	10	T01
	8	T05
	3	T08, T14
	2	T02, T06, T12, T16, T19
	1	T04, T07, T10, T11, T13, T15, T18, T20, T22, T28

provide Table 2 for the demographics of SCA tools used by participants.

4.1.1 Reason for using SCA tools

Most (15) of the participants explicitly mentioned security as the main reason for running SCA tools. When using SCA tools for security, the main goal is to find and manage vulnerabilities. Participants want to make sure that the open source libraries they used do not include vulnerabilities. However, security was not the only reason mentioned for using these tools. About half (10) of the participants also explicitly mentioned compliance and licensing issues as a reason for using SCA tools. Software has an impact on many areas. For example, automobile software has to adhere to regulations that govern the automotive industry. Hence the need for software compliance.

Some (3) participants also mentioned less intuitive reasons for using SCA tools, including merger acquisitions and export relations. These use cases might be less obvious to a engineer, but are also an important use for SCA tools.

> "*You have license compliance, security, export restrictions. AKA, I cannot have code from any of the banned countries. [...] For export compliance, I cannot have U.S. laws very strict on certain cryptographic algorithms. [...] So, we're buying a company or we're being bought. And you basically have to prove that your stack, if your stack is somewhat decently built.*" - P08

4.1.2 Selecting SCA tools

There were multiple factors that influence the selection of SCA tools. A few participants said that their organization was already using the same tool. Another factor mentioned by some participants was ease for deploying the tool. Platforms such as GitHub provide SCA features in their tool and allow engineers to easily integrate it into their projects hosted on the platform. "*It's much easier to have it all inbuilt in the same platform rather than using external tools, which were required to set up integrations and configurations.*" (P05)

Challenges with tool deployment or tool reports led some (5) participants to report switching tools. Participants reported using one SCA tool initially, but encountered issues and challenges, for example: tool requirements were hard to fulfill "*The scan requirements to run T05 were very difficult to fulfill for some languages and for the varying build processes that we used.*" (P03) or they were not satisfied with the tool's results. "*It was not uncommon that the database would be reporting something wrong [...] it would match a component to an incorrect component, or it would report a license. And that product had not been built with DevOps and CI/CD in mind.*" (P09) This led them to switch to another SCA tool that addresses their challenges. A few participants also mentioned using multiple tools, because the different features in different tools complement each other. "*I use combination of tools actually, since some of those tools are free. I use majorly T09 (OsT) and second is T12 for getting results instantly. [...] T09 (OsT) is very handy for generating reports.*" (P13)

> Summary: SCA Demographics. The main reason for using SCA tools are vulnerability detection and license management. Factors impacting the selection of the tool vary among participants, including ease of deployment and result accuracy. It was also common to switch tools after encountering challenges with the first tool.

4.2 SCA Integration

The next main topic in our interviews was SCA integration. We asked participants how they ran SCA within their organizations. This was key to understanding the entire SCA process in development pipelines and provides better context for understanding challenges.

4.2.1 Input to SCA analyses

SCA tools use three major types of input: source code, binaries, and software metadata. Different types of analyses tailor to users with different use cases and priorities.

Source code and metadata analysis: Source code analysis was the most common; however one concern brought up by some participants was the leaking of intellectual property, i.e., the source code. "*We didn't use the cloud version because we didn't want the code to go outside.*" (P05) SCA tools that only analyze metadata present less risk and can address concerns about leaking intellectual property. "*One of the benefits of our product is that we only look at manifest, because giving your code base up to a third party tool is kind of risky sometimes.*" (P04)

Binary analysis: With source code analysis, the tool is able to identify software components from manifest files or metadata. This information is not available to binary SCA analysis.

Therefore, binary SCA analysis must rely on techniques such as fingerprinting to identify components. Some (3) participants shared frustrations with identifying components and versions in binaries. "*So a lot of the tools are very good at saying open source code is present. So like identifying that OpenSSL, just to pick a random example, is in the code, but they're less good at identifying a specific version of OpenSSL.*" (P11)

With fingerprinting, if a single line is changed in the file, the fingerprint is completely different. "*If you change one single line of code [...] you might get a different binary [...] Therefore, you will not have a match.*" (P08) Determining what information to focus on, whether to fingerprint an entire file or only a function can be a challenge for binary analysis. Furthermore, tools may not be focusing on the important information within the binary.

> "*They're indexing too much. They're not looking at what kind of files are we actually looking at. A good example is when you're scanning something like a .NET project. [...] you have lots of generated code like interfaces for the user interface. And that's just a lot of generated code where the tools that create the user interface, when you generate them, they just output a lot of boilerplate.*" - P12-dev

4.2.2 SCA in the software development lifecycle

SCA can be integrated at various stages in the development lifecycle. We discuss how SCA tools are integrated both within and outside of CI/CD pipelines.

SCA inside CI/CD pipelines: Most (16) of the participants shared that they integrated the SCA scan into their CI/CD pipelines. This could be in the form of GitHub actions run during merge requests or pull requests. It could also be other custom pipelines used by organizations, e.g., Jenkins. The majority of (13) participants mentioned the SCA is triggered by code commits, merge requests, or pull requests. Participants also pointed out, as the SCA tool is integrated into the build pipeline, it can disrupt software deployment.

> "*There is a build pipeline that takes this code after the code is merged to the GitHub repository, performs different checks, including security checks, and then creates a build and optionally deploys it to the target platform or makes it available for users. [...] if there is a security issue, the SCA tool may say, oh, you have an issue, please fix it and rerun the pipeline again.*" - P15

Some participants mentioned other ways an SCA tool could be integrated. Some vendors allow users to execute an API call to the vendor's backend to run an SCA scan. Other participants also shared using the SCA tool as a standalone application.

SCA outside CI/CD pipelines: Similar to unit tests, SCA scans are run frequently in the build pipeline. However, SCA are also run before and after build and deployment. Some SCA vendors allow users to directly run the SCA feature in their integrated development environment, for example as a VS Code extension. A participant pointed out that integrating SCA in multiple stages and early in the development cycle can be beneficial. "*We make T05 available to our development team so they can use it as they need to throughout development.*" (P11) Finding vulnerabilities early on was beneficial for them as vulnerabilities can add up during their multiple month development cycles. "*Our development cycles can last months and we don't want to wait until the last minute to figure out, hey, we need to change major components.*" (P11)

For more complete monitoring, deployed applications also need to run SCA frequently due to new vulnerabilities being found. SCA results can change after the application has finished development, as vulnerability results can be updated, further adding to the need to run SCA frequently. "*Sometimes when you go to the NVD, you'll see something say undergoing reanalysis.*" (P09) One participant reported that their customer would run an SCA scan on the application after deployment and receive different results "*I had a customer reported thing in an older version of one of our pieces of software*" (P09) This case required back and forth with the customer to resolve the issue.

4.2.3 Challenges when integrating SCA

Many challenges emerged when we discussed integration of SCA with participants.

Challenges with different ecosystems: Some (6) participants pointed out that SCA tools often perform better for some ecosystems than others. Strengths and weaknesses are different for each tool. "*Both T05 and T06 were pretty good at certain languages. I think T05 was really good at Java, and probably Python. [...] they were both pretty bad at Go.*" (P03) Participants sometimes had to work around the weaknesses or select another tool. "*We had to build a lot of, kind of additional tooling to help remove false positives for Go projects.*" (P03)

One of the participants with SCA development experience offered some insight into the reasons for this. "*All the package managers are different. So you have to have exceptions for each one and you have to figure out what those exceptions are.*" (P04-dev) Ruby has a lock file that locks the library versions. Python can be trickier because there are different lock file types and multiple ways of resolving dependencies. JavaScript has the issue of having too many layers of dependencies, cycle may be present too. Maven does not have lock files. "*It's an older package manager. They have the pom file, which is your main manifest. And then you can have parent poms or other types of poms, and you have to resolve those. [...] And then there's also interpolation. So you might have manifests with interpolated versions and you have to figure out what that is.*" (P04-dev) Gradle does not have a static

manifest so it could be very difficult to resolve dependencies. "*They're actually scripts. You could have like a production and a test branch in your Gradle script, but you could also have custom scopes [...] they can be nested in folders and it's very complex.*" (P04-dev) On top of all the complexities, SCA maintainers also have to keep track of changes in the ecosystems or package managers.

Unsupported scripts: Some participants mentioned challenges with scripts not supported by the SCA scan. These included custom build scripts in the software or legacy programming languages, such as Lisp or Fortran. "*There is some software written in these languages as well. Or some exotic languages maybe. And there is simply not that much knowledge and expertise to create a product that they can analyze and successfully.*" (P15) When a file format is not supported by an SCA scan, users have to find other ways to run the scan. "*We need to go and either pre-process things ourselves. We need to work with the supplier to add support for new file formats.*" (P11)

SCA halting the development cycle: Integrating SCA scans into the build pipeline could act as a double edged sword. Some (6) participants mentioned SCA scans can bring software development to a halt. Participants mentioned SCA alerts failing builds or the the SCA tool having issues itself. Allowing SCA tools to fail builds means the development cycle is halted until the issue is fixed. "*They slow down velocity [...] like everything just comes to like a screeching halt until we, until we fix it. We can't like merge anything and we can't run tests and stuff like that.*" (P07) One participant shared that due to these failed builds, they transitioned to running the SCA scan every week. "*Previously, we had it set up with a custom action that like, when you push the code, it would do the scanning. Now we have like scheduled it. It's going to run every week automatically once to scan everything. We don't have it on each push anymore [...] It doesn't become a blocker.*" (P05)

Failed builds slows down the development process because engineers have to stop development to figure out why the builds are failing.

> "*What happens more than blocking, like actually blocking builds is the sense that it's blocking work. Where developers will see a failed check. Technically they can still merge it if they want. [...] they try to do the right thing or understand why something's failing, and so it causes at least a detour where they have to like reach out, you know, in our like Slack. [...] So you get a temporary implicit block. Um, usually not for super long, but enough that, you know, for every developer that's kind of annoying.*" - P16

Using multiple tools: Some participants shared that they use multiple tools to gather a more holistic picture. For one of the participants, a combination of different tool results provides a more holistic view of the application. The participant has software in containers, and software in the container is built from microservices. They have SCA scans at both the microservice and container level. The different results complement each other because they provide results at different levels vertically along the development lifecycle.

> "*They definitely complement each other. The microservice level is for, we kind of think of things as first degree dependencies or third degree dependencies. If my microservice is pulling this in and I'm not getting it from platform, then I'm the one who's responsible for monitoring its security. And so for that reason, we need to have that microservice view.*" - P09

However, another participant pointed out that integrating multiple tools into their software pipeline can be a huge challenge. Differences in tools require additional wrappers or scripts to connect the tools, and with it comes additional effort required for the maintenance of the entire pipeline. "*Integration is near impossible. [...] They pretty much built every SCA tool at the time of the market and they spent millions to build a Frankenstein monster to connect all of those SCAs together.*" (P08) Even with integration, the upkeep of the tool could be a huge cost.

> "*SCA, what it's detecting is the ecosystem, the Java ecosystem, the Node ecosystem, those continuously change. [...] new build tools could vary. And there are also multiple new build tools coming on the market. If your SCA tool doesn't support those build tools, you're screwed. So, the upkeep of that tool is basically, it's not sustainable or maintainable.*" - P08

> Summary: SCA Integration. SCA tools are most often integrated into build pipelines and run on pull requests. While CI integration automates SCA scans and catches vulnerabilities, it can also block development pipelines. Huge efforts are needed if trying to integrate multiple SCA tools together.

4.3 SCA Tool Report

The third main topic discussed in our interviews relates to actions after running the SCA scan. We wanted to understand how the outputs are interpreted and processed, and further, how the vulnerability alerts are resolved. In this subsection, we discuss: (1) how SCA tool outputs are interpreted, (2) how vulnerabilities are fixed, (3) false positives, and (4) tooling around the SCA outputs.

4.3.1 Interpreting vulnerability reports

The majority of of our participants have a role title related to security. Part of the workload for these participants was to focus on the SCA process and outputs. These security

engineers reach out to the engineers who wrote the code to discuss a solution for vulnerability alerts. "Tickets" are commonly used by security engineers to relay this information to engineers. "*I will look at the findings from the SCA tool and triage them and determine which ones need to be tickets that would then go to the engineering team for remediation.*" (P06)

The number of SCA alerts can initially be overwhelming. Most (15) participants mentioned prioritizing alerts based on severity. Severity is either CVSS score or a metric specific to the SCA tool. Other mentioned prioritization methods include: (a) the time required to fix the vulnerability and (b) whether the application is of importance. After initial prioritization, participants evaluate the risk and impact of the vulnerability. Various factors come into play, including: (a) whether the vulnerability is exploitable and (b) whether the application is external facing or behind a firewall. This evaluation process requires more context and thus security engineers often have to review each vulnerability manually.

> "*Ideally as a security team, we know what the product is. So you'd be able to tell, like, if you see a UI, you know, hopefully you might be able to tell like which feature it might be in [...] Then maybe, you know, which feature that would go in and that would help you determine, you know, how impactful that specific vulnerability might be.*" - P06

Many (8) participants explicitly expressed the need for additional context in vulnerability alerts. One participant gave an example of how additional context greatly helps with determining how to resolve vulnerability alerts from the tool. "*After getting all the vulnerabilities. [...] you will get one alert from SCA tool and one from SAST, one more from AWS or Infraside. This tool will combine three of those outputs and tell you if these three can be combined, become exploitable thing or not.*" (P13)

One participant noted that sometimes not being able to explain the results from the tool can be a challenge. "*I really want to verify because if, when I'm talking to lawyers, I want to be able to explain like, this decision was made and this is why.*" (P12)

4.3.2 Fixing vulnerabilities

Updating a library to a newer, non-vulnerable version is the most common method of resolving a vulnerability, as mentioned by the majority of (13) participants. However, participants also run into cases where updating the library can cause compatibility issues within their application. Many (8) of the participants reported that sometimes updating dependencies would introduce breaking changes into the application. "*Let's just bump it up. So it goes away, and I hope that nothing happens. But there is on occasion, that you would introduce a breaking change if you were to up the version.*" (P06) When a simple solution does not exist, more investigation is needed to come up with a fix for the vulnerability. Sometimes, it required a great amount of additional effort, as explained by one of the participants. "*It was going to be very, very time intensive to do a rip and replace. So instead, we forked it and we took out the vulnerable pieces. So our fork didn't have the vulnerability. [...] we took that class out or we took that method out.*" (P09)

Exceptions: Some (4) participants explicitly mentioned that they had an exceptions process in place for resolving vulnerabilities. In the exceptions process, the engineer manually reviews the vulnerability and provide reasons why the vulnerability does not need to be removed. Examples of an exceptions include: (a) the vulnerability does not impact application (considered a false positive) and (b) the business risk from removing the vulnerability does not outweigh the security risk it brings.

> "*This thing needs to ship or we don't make money. Can we get a security exception? [...] you have to weigh what's the security risk versus what's the business risk. And at some point, it's a choice between the business going belly up and potentially getting breached. That's a call that upper management has to make.*" - P10

Isolating vulnerable components: Participants also mentioned isolating the vulnerable component. Oftentimes, removing network connections to a component eliminates risk, because no external input can exploit the vulnerability. "*Most security vulnerabilities, the risk of that can be negated if you basically just make sure that it doesn't have a connection to the internet.*" (P08) By isolating the component, the risk of the vulnerability reduces to an amount that is small enough to be accepted. This is not the ideal solution but can be used as a last resort. "*There have been cases where we might accept risk of a vulnerability, but only for a certain amount of time. [...] and what you would do is you would do a limited release, but you would say, okay, you need to add these firewall rules so this vulnerability can't be exploited.*" (P09) One participant explains that there are times when a non-vulnerable alternative is not available, thus the only choice is to make sure the component is not exposed. "*We've got some suppliers that supply industry. They are the only choice through the dominant force. It's basically you've got a choice of either you work with them or you don't have a product in that area.*" (P11)

4.3.3 False positives

In our study, we consider a false positive to be an SCA vulnerability alert where the vulnerability does not impact the software. False positives are a big concern for SCA users, a topic discussed by all except one participant (15). One participant explicitly stated false positives as the reason they switched from on SCA tool to another. "*[T03 (OsT)] throws a lot of false positives. That's the main reason that we adopted*

the commercial tool" (P02) The reasons for false positives differ. One reason mentioned was test or build dependencies. "*Some of it was just like the tool couldn't ignore test dependencies.*" (P03)

While *impact* could be defined differently for different engineers, almost all the participants agree that a lot of the SCA vulnerability alerts do not pose a direct threat or impact to their application. "*I do think that many SCA findings do not present an actual risk to an application.*" (P06) Determining the actual impact of the vulnerability requires experience and knowledge about the vulnerability and the application itself. This process is not easily automated and requires manual effort. Often, the security team and the engineering team need to work together to determine whether an alert is a false positive. "*[security team] they're the ones responsible for like investigating a CVE [...] And working with the development teams to answer: Is this a false positive? Is it a true positive?*" (P10)

An approach to combat false positives mentioned by one of the participants was to include library maintainer insight into the vulnerability alerts, as library maintainers are the ones most familiar with the code in the library. "*Those maintainers give reports about CVEs on their projects. So they, the maintainer can tell [...] that a certain CVE is a false positive. And so we can surface that to our customers.*" (P04)

4.3.4 Tooling and automation around SCA output

As discussed in Section 4.3.1, additional manual effort is often needed after receiving SCA reports. Some (4) participants shared how they built automation and tooling around the SCA tool to reduce the effort needed to review vulnerabilities. One of the main advantages mentioned was the tooling reduced efforts to fix the same issues over and over. The tooling would be able to resolve previously fixed vulnerabilities. If a single vulnerability affected multiple applications, the tooling would also link all the results to a single result, which eliminates the need to fix duplicate issues.

> "*With earlier tooling, each software component that used a fundamental library like Spring Framework would have had an individual ticket for a critical vulnerability, but because they ingested it from a base image, absent forking, they could not directly remediate the issue themselves. We [...] have [evolved] the automation [to] find the root cause [...], which component or project is initially bringing this in. And then, everybody who's getting it transitively, they still have a ticket, but their ticket will link to the root cause.*" - P09

The tooling sometimes also learns from past feedback and experiences from the user and applies that on newly found vulnerabilities. This capability eliminates the need for engineers to re-evaluate the same issues.

> "*We build tooling around it that interprets the results [...] lets us mark it in version 1 that says this isn't applicable because it doesn't apply to our architecture. And then on version 2 of the product, when we scan it with T05 and get the same finding, the past triage [...] is automatically applied to it [...] it basically carries forward the learnings and the scorings from past investigations into vulnerabilities and applies those to the same thing for the same code base for later versions of it.*" - P11

Unknowns in how an SCA tool worked also presented challenges for users. For example, debugging issues with a tool requires knowledge of how a tool works. Without an understanding of how SCA tools run, users are not able to fix issues encountered when using the tool. A few participants discussed unknowns in the tool as a challenge. "*There were a lot of unknowns in how the product worked. And so we would use it a certain way and then it might crash, or it couldn't handle the number of scans.*" (P03)

Different approaches between small and large organizations: We observed different approaches of interpreting and acting on SCA results between small and large organizations. Larger organizations have a larger workforce, thus they can allocate more effort into specifically building and maintaining a tool around the SCA process that tailors to their pipelines. "*My team develops the security automation tools. And then we have a security team who's a little bit more of the higher level of security. And they're the ones responsible for like investigating a CVE.*" (P10) Smaller companies are not able to do this and the developers often have to take care of the SCA process from start to finish. There is also less testing in place to catch issues with updating library versions. "*I think many companies don't have that maturity in their end in testing. And so, you know, this, it's risky to say, okay, let's bump up this JavaScript library up to here and hope everything works out.*" (P06)

> Summary: SCA Reports. False positives from the tools are common, and users are not able to determine the impact of the alert solely from the tool reports. Manual effort is needed to gather additional context relating to the vulnerability and assess the impact of it. In an effort to reduce manual labor, organizations are also spending additional effort developing tooling around SCA tools to manage the large number of vulnerability alerts. Sometimes fixing the alerts is simple as updating the library version. Other times, additional effort is required to propose a solution that does not break the application and fixes the vulnerability. Resolving vulnerabilities differs case by case.

4.4 Developer Opinions

In our interviews, we also gave participants the opportunity to voice their opinions about SCA tools and vulnerability data in general. In this subsection, we discuss several opinions from participants.

False sense of security: Some (3) participants reported that SCA can only identify known vulnerabilities and can give you a false sense of security. Running an SCA scan can give the impression that the tool will report all vulnerabilities. However, there could be existing vulnerabilities that have yet to be reported publicly, those vulnerabilities can pose a huge risk to organizations. "*SCA scans are great at pointing out vulnerabilities that people have already found, but they're not going to help you with vulnerabilities that people haven't found. [...] But it's the threats that you can't see that will kill you.*" (P10)

CVE data: Some (3) participants expressed their opinions on how CVEs are reported and managed. They expressed concern over the large number of CVEs and the need for better CVE standards. As discussed in Section 2, CVE data does not have specific requirements other than the ID, description, and references. "*There are just so many CVEs being submitted all at once that there's just no way to properly govern them all. [...] I'm exaggerating, but you can submit like a children's novel and get published as a CVE. I think that the standard for submitting a CVE can definitely go up, can definitely improve.*" (P06) A participant pointed out that the metrics used to convey vulnerability severity can be misleading. "*[CVSS score] is a big lie, in my opinion. CVSS scores. The people who report it to NVD, they try to inflate it most of the time. So it's not at all helpful.*" (P13)

4.5 Improvements for SCA

In the last topic of our interviews, we asked participants what kind of improvements they would like to see.

Context: The most commonly discussed improvement was additional context for more actionable actions for SCA users, as discussed in Section 4.3.1. One participant proposed an approach that combines SCA, SAST, and infrastructure context. This approach greatly helped them with resolving SCA alerts. "*SCA itself is not that useful since we do not have full picture. So SCA combined with SAST is very helpful. And if you add your infrastructure configuration context also, it becomes a lot more helpful to prioritize things.*" (P13) Some participants also pointed out that LLMs could help by providing more information on vulnerabilities.

Reachability: Reachability was another concept brought up by about half (9) of participants. "*A lot of these findings that come from these SCA tools, are of, you know, functions that don't ever get called. And so therefore they don't present any risk, confirmed risk to the application.*" (P06) In the case of vulnerabilities and SCA, reachability is often used to analyze whether a vulnerable function is called from the main application. It can help by narrowing down vulnerability alerts to the functions that are actually called. It is a fairly new approach adopted by some SCA tools, but some (4) participants believe it can greatly improve to reduce efforts needed to manage alerts from SCA tools. "*I think the reachability analysis would be the biggest game changer that will come to SCA in the next five years.*" (P06)

User Feedback: Another potential improvement brought up by some (3) participants was some kind of feature in the tool that would receive user input and improve future results based on that. The motivation comes from tool results being inaccurate or irrelevant sometimes, and the user may want to override results or teach the tool to make different determinations for future results. This concept is similar to the motivation for the additional tooling around an SCA tool discussed in Section 4.3.4. "*Having more intelligence to them where you can better train them to say, hey, this isn't an issue for us in this context. So don't flag this as an issue next time around. So basically eliminate the need for some of the custom tooling we've written.*" (P11)

Better License Detection: SCA relating to managing licenses was brought up by the majority of participants, as discussed in Section 4.1. Some (3) participants explicitly expressed opinions for better detection and management of licenses. One participant provided some insight as to why license detection can be a challenge. "*For many licenses, there is a standard way to basically say this file is under this particular license. But there are lots of stubborn developers who are thinking like, no, I'm going to do it in a different way because that's better. And then for the scanning tools, basically it causes issues.*" (P12)

New Metrics: Lastly, A few (2) participants expressed the need for the tools to be more proactive to adapting new metrics and having more support from the tool or the community around the tool.

> Summary: Improvements. Information in SCA alerts are too generic for users and lack context. Participants wanted more specific information on actions they can take and insights into the impact of the vulnerability on their application.

5 Discussion

In this section, we summarize our findings by (1) presenting Figure 5, a revised version of Figure 1 that includes our findings on how SCA is integrated into development pipelines and (2) answering our research questions and discuss how each finding relate to our main finding: **context matters**. More context is needed throughout the SCA process to streamline vulnerability management. We identified three emergent themes through which context matters:

- **T1:** More information is needed to evaluate the risk and impact of the vulnerability alert.
- **T2:** When and how the SCA scan runs in the development cycle matter.
- **T3:** Integration of SCA and communication of the results

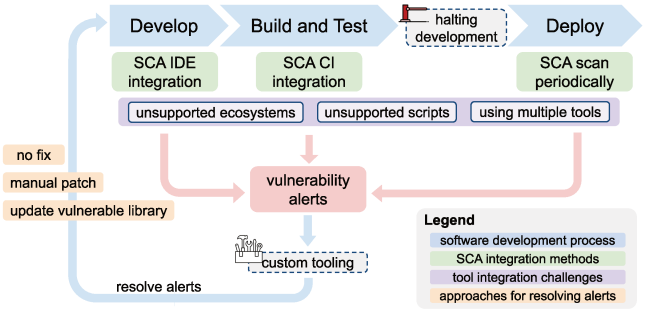

Figure 5: SCA integration in the software lifecycle including our findings on how SCA is integrated into development pipelines.

can generate overhead if there is not enough context present.

While the SCA industry is beginning to promote reachability analysis to determine if a vulnerability impacts a project (a form of context), our study reveals the need to consider context more broadly. In the following we detail how the identified themes related to our findings.

Answering **RQ1**: *How do users interact with SCA tools?* Participants mentioned using SCA tools as standalone applications or integrating them into CI/CD pipelines, such as through GitHub Actions. After receiving vulnerability alerts from an SCA tool, engineers can resolve it themselves or forward the issue to the responsible engineer. Security engineers often need to work together with software engineers who have a better understanding of the code and come up with a solution together. Security engineers lack the context software engineers have of the code and therefore need to coordinate with them (**T3**). Participants reported going through the alerts manually to determine how to resolve them (**T1**).

5.1 Challenges with SCA Deployment

Answering **RQ2**: *What are the challenges when deploying SCA tools?* Many challenges emerged when we talked to participants. We grouped the challenges into corresponding themes and discuss them individually.

Tradeoffs that come with integrating SCA in CI/CD (T2): The *when* and *how* of an SCA process should be customized based on the team's workflow and priorities. Running an SCA tool as-is without consideration of how it fits into pipelines presents challenges. Integration of SCA tools through CI/CD pipelines automates the scans and takes the burden off engineers. However, SCA scans can halt build pipelines when they are used as gates to prevent insecure code from being merged into codebases. This can be a blocker in unintended situations; a merge request blocked due to a vulnerable library can delay the development of a new feature, even when that library is completely unrelated to the feature. In Section 4.2.2, we discussed an example where the team opted to run SCA scans themselves every week instead of integrating it in the CI pipeline.

SCA scans need to be run frequently (T2): Unlike SAST tools that return the same results for the same codebase on reruns, SCA scan results can change overtime due to the discovery of new vulnerabilities in components while the codebases remains the same. Thus, SCA needs to be run frequently, even after software has been deployed. Running scans frequently also prevents a large amount of alerts overwhelming engineers with each scan.

Balancing SCA in CI/CD (T2): Based on our interviews, a common way to run SCA scans frequently is to integrate them into CI/CD pipelines. However, if not configured correctly, SCA scans can halt development pipelines. SCA alerts can be noisy, thus build pipelines would halt frequently if they are blocked on all SCA alerts. The balance between running SCA scans and keeping the development pipeline flowing is a challenge, similar to the tension between security and usability. Users should consider (1) the type of vulnerabilities to prioritize and only blocking builds on those and (2) the additional effort to fix an alert blocking the pipeline. The balance of both ensures a secure build and a usable SCA integration.

Overhead in the SCA process (T3): Ideally, the SCA process is fully automated to facilitate continuous scanning runs. Unfortunately, SCA tools sometimes run into edge cases that the tool cannot handle, e.g., unsupported file formats and different ways to import libraries. To resolve the edge cases, engineers need to put in extra effort. The overhead increases exponentially when up-scaling SCA process in large codebases or organizations. Related, different teams may use different software and technologies, making integrating and scaling SCA processes across multiple teams a challenge: "*I know for certain of their product teams, they use six or seven different SCA tools. [...] Every team has a different workflow, different configuration. Now, do this at scale.*" (P08)

5.2 Acting on SCA Results

Answering **RQ3**: *What are the challenges when acting on SCA results?* From our interviews, we observed that some organizations have automated tooling present to assist with managing vulnerability alerts, some do not. Regardless, vulnerability alerts require manual review to determine risk and impact to the application (**T1**). Various approaches can be taken to resolve the alerts, the most common and easy way is to update the library to a newer or non-vulnerable version. However, sometimes updating library versions introduce breaking changes into the application. If there is no easy way to update the library to a non-vulnerable version, exceptions such as accepting security risk or isolating the vulnerable component to mitigate risk are made.

More context needed for vulnerability alerts (T1): A common complaint from participants was the lack of context within vulnerability alerts. SCA tools output all vulnerabilities found in libraries used in the application, regardless of how the library is used. Participants mentioned that the vulnerability often does not affect their use case. Furthermore, the alerts are often generic and do not provide enough information on how to resolve the vulnerability. Users often have to find more information on the vulnerability to determine how to resolve it.

Fixing SCA alerts at scale (T3): The number of SCA alerts at a large organization overwhelms security engineers. As discussed in Section 4.3.4, large organizations invest significant effort into building automated tooling around the SCA tool outputs in order to manage them efficiently. Multiple SCA alerts may be related to the same root cause. Rather than each engineer resolving the alert individually, the tooling eliminates duplicate efforts. The automated tooling relays context along the SCA process, but at a large maintenance cost for organizations. Challenges also arise between software vendors, customers, or auditors when the different parties receive different SCA results: "*If your customer or an auditor is scanning with a different tool, they might get different results. And the results might be not just the library of the CVE, but the severity can differ.*" (P09)

5.3 Improvements for the SCA process

Answering **RQ4**: *How can the SCA process be improved?* Based on previous discussion and results, we suggest several approaches for stakeholders to address the lack of context throughout the SCA process.

Tool users: For SCA users, finding a tool that fits well into the engineering teams' software pipeline is beneficial (**T2**). No tool is perfect, and understanding the strengths and weaknesses of each tool helps with tool selection. As discussed in Section 4.2.3, SCA tools can perform differently for different languages. Some tools require access to source code or binaries, others only require metadata; this could be a concern for companies that value the confidentiality of their source code. Different tools may also have different integration methods into the software development pipeline. These should be taken into account when selecting SCA tools. Furthermore, not everything in the SCA process is currently automatable. Resolving vulnerabilities often requires multiple teams to work together. Understanding the tool and having clear communication between engineering teams will make the process smoother (**T3**).

Tool vendors: For SCA vendors, more information about vulnerability alerts is greatly needed (**T1**). SCA users often require in extra manual effort to determine how to resolve the alerts. Providing less generic information, and more information specific to the users' use case can reduce this effort, this could include the following information:

- **Function reachability:** Considering how the vulnerable dependency is used within the application and whether the vulnerable function is reachable through call graphs would help filter out unreachable vulnerabilities and reduce false positives, as discussed in Section 4.5.
- **Infrastructure:** Different applications are set up differently. Understanding the infrastructure underlying the application and how the application interacts with users and other applications will help identify how data is passed through the application, addressing Section 4.3.1 and 4.5.
- **Network configurations:** As discussed in Section 4.3.1 and 4.3.2, vulnerabilities are often exploited through untrusted data input. Some applications sit behind a firewall that protects it from untrusted data, hence the application would not be vulnerable to attacks through untrusted input data.
- **Exploitability:** Combining the points listed above with SAST capabilities and providing information such as data flow analysis or sanitization will provide developers with a bigger picture (Section 4.3.1), with which they can make better decisions to determine the exploitability of a vulnerability. SCA vendors could also consider adding LLM inference to combine all the information gathered to provide SCA users with a clear suggestion on resolving the vulnerability.

Furthermore, participants also desired a feature that allows users to train the tool or provide feedback on results. SCA tools can also be improved to address process overhead (**T3**). For example, participants created custom tooling around SCA tools. Vendors should consider how to make SCA platforms more customizable to different pipelines.

Researchers: We encourage researchers to investigate techniques to (a) provide more context about the vulnerability and its impact and (b) reduce false positives (**T1**). As we discussed in Section 4.5, participants mentioned alerts from SCA tools being too generic. Techniques such as debloating and reachability have potential to identify if a vulnerability impacts an application. Research on how to assist SCA tool vendors in providing better quality data for the four types of information mentioned can help provide better context. Research on how to use LLMs to reduce manual effort of interpreting SCA results would be beneficial. LLMs could be used to (1) combine the four types of information to determine vulnerability impact and (2) provide users with fix suggestions.

6 Conclusion

To understand the use of SCA tools and challenges that come with it, we conducted 20 semi-structured interviews with industry professionals with SCA experience. The interviews uncovered challenges with integrating the tools and

challenges with resolving the vulnerabilities outputted by the tools. We found that throughout the SCA process context matters and that the lack of it creates challenges for the users.

Acknowledgments

This work was supported in part by the National Science Foundation grant CNS-2207008. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies. We thank our participants for participating in the study, sharing their experiences, and providing valuable input. We also thank all the anonymous reviewers and shepherds for their thoughtful comments and feedback.

Ethics Considerations

We structured our interview study to follow the ethical principles outlined in the Menlo report. Prior to conducting interviews, we obtained approval for our study setup by our university's Institutional Review Board (IRB). We provided all invited participants with detailed information on our study's goals, methods, and data handling strategies. We sent out a consent form to inform each participant of study details, how their data will be handled, and their freedom to skip any question or withdraw from the interview at any time. Before each interview, we obtained informed consent from each participant and encouraged invited participants to bring up any questions or concerns they have and addressed their concerns.

All data was collected and stored according to our IRB guidelines and GDPR. All authors went through IRB training for data handling. Participant data, interview recordings and transcripts were stored locally in an encrypted volume. Audio recordings of the interviews were turned into pseudonymized / redacted transcripts using a locally-run OpenAI whisper machine learning model and recordings were deleted after transcription.

We informed the participants of the risk in participating in our research in our consent form. The primary risk in our study is reputational damage to the participant or their organization when describing their security practices. To prevent damage, we stored identifiable data in encrypted volumes and removed all identifiers in publicly available data. As compensation for sharing their experiences and valuable time for 45 to 60 minutes, we offered each participant $60 or the equivalent value in Amazon.com vouchers. The benefit of our study is motivation for stakeholders to improve the SCA process. We weighed the risks and benefits of our study and believe that the benefits outweighed the risks.

Open Science

We acknowledge the USENIX Security open science policy. The research artifacts associated with this study are:

- Raw interview transcripts
- Anonymized interview transcripts
- Interview guide
- Codebook

Things we have shared: The interview guide is an important part of the design of our interview study, which is included in Appendix A. We also share the codebook, with codes and example subcodes, in Appendix B. We discuss identified major themes in Section 5. We have also shared an online replication package including (1) the consent form, (2) the interview guide, and (3) the codebook. The replication package can be found at https://doi.org/10.5281/zenodo.15537121.

Things we cannot share: We cannot share recordings as these were destroyed after transcription. For the privacy of the participants and compliance with IRB, we cannot share raw transcripts. We are also unwilling to share the anonymized transcripts as there is a high risk of de-anonymizing the participants given the large amount of discussion in the interviews.

References

[1] *2024 Open Source Security and Risk Analysis Report*, https://www.blackduck.com/resources/analyst-reports/open-source-security-risk-analysis.html, 2024.

[2] M. Alfadel, D. E. Costa, and E. Shihab, "Empirical analysis of security vulnerabilities in python packages," *Empirical Software Engineering*, vol. 28, no. 3, p. 59, 2023.

[3] A. S. Ami, K. Moran, D. Poshyvanyk, and A. Nadkarni, ""False negative - that one is going to kill you": Understanding Industry Perspectives of Static Analysis based Security Testing," *2024 IEEE Symposium on Security and Privacy*, 2024.

[4] A. Anwar, A. Abusnaina, S. Chen, F. Li, and D. Mohaisen, "Cleaning the NVD: Comprehensive quality assessment, improvements, and analyses," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4255–4269, 2021.

[5] H. Assal and S. Chiasson, "Security in the software development lifecycle," in *Fourteenth symposium on usable privacy and security (SOUPS 2018)*, 2018, pp. 281–296.

[6] H. Assal and S. Chiasson, "'Think secure from the beginning' A Survey with Software Developers," in *Proceedings of the 2019 CHI conference on human factors in computing systems*, 2019, pp. 1–13.

[7] M. Bailey, D. Dittrich, E. Kenneally, and D. Maughan, "The Menlo Report," *IEEE Security & Privacy*, vol. 10, no. 2, pp. 71–75, 2012.

[8] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker, "Field studies of computer system administrators: analysis of system management tools and practices," in *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, 2004, pp. 388–395.

[9] G. Bhandari, A. Naseer, and L. Moonen, "CVEfixes: automated collection of vulnerabilities and their fixes from open-source software," in *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2021, pp. 30–39.

[10] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung, "When and how to make breaking changes: Policies and practices in 18 open source software ecosystems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–56, 2021.

[11] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[12] *CVE Metrics*, https://www.cve.org/about/Metrics.

[13] A. Dann, H. Plate, B. Hermann, S. E. Ponta, and E. Bodden, "Identifying challenges for oss vulnerability scanners-a study & test suite," *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3613–3625, 2021.

[14] J. Dietrich, S. Rasheed, A. Jordan, and T. White, "On the security blind spots of software composition analysis," in *Proceedings of the 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2023, pp. 77–87.

[15] T. Dunlap, E. Lin, W. Enck, and B. Reaves, "VFCFinder: Seamlessly pairing security advisories and patches," *arXiv preprint arXiv:2311.01532*, 2023.

[16] *EO 14144: Strengthening and Promoting Innovation in the Nation's Cybersecurity*, https://www.federalregister.gov/documents/2025/01/17/2025-01470/strengthening-and-promoting-innovation-in-the-nations-cybersecurity, 2025.

[17] P. I. Fusch Ph D and L. R. Ness, "Are we there yet? Data saturation in qualitative research," *The Qualitative Report*, vol. 20, no. 9, pp. 1408–1416, 2015.

[18] M. Gutfleisch, J. H. Klemmer, N. Busch, Y. Acar, M. A. Sasse, and S. Fahl, "How Does Usable Security (Not) End Up in Software Products? Results From a Qualitative Interview Study," in *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*, IEEE Computer Society, May 2022.

[19] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *network and distributed systems security symposium*, 2019.

[20] N. Imtiaz, A. Khanom, and L. Williams, "Open or sneaky? fast or slow? light or heavy?: Investigating security releases of open source packages," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1540–1560, 2022.

[21] N. Imtiaz, S. Thorn, and L. Williams, "A comparative study of vulnerability reporting by software composition analysis tools," in *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2021, pp. 1–11.

[22] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" In *35th IEEE/ACM International Conference on Software Engineering (ICSE'13)*, IEEE, IEEE, 2013, pp. 672–681.

[23] A. Krause, H. Kaur, J. H. Klemmer, O. Wiese, and S. Fahl, ""That's my perspective from 30 years of doing this": An Interview Study on Practices, Experiences, and Challenges of Updating Cryptographic Code,"

[24] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration," *Empirical Software Engineering*, vol. 23, pp. 384–417, 2018.

[25] V. B. Livshits and M. S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis.," in *USENIX security symposium*, vol. 14, 2005, pp. 18–18.

[26] *Log4j - CVE-2021-44228*, https://nvd.nist.gov/vuln/detail/cve-2021-44228.

[27] N. McDonald, S. Schoenebeck, and A. Forte, "Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice," *Proceedings of the ACM on human-computer interaction*, vol. 3, no. CSCW, pp. 1–23, 2019.

[28] S. Mirhosseini and C. Parnin, "Can automated pull requests encourage software developers to upgrade out-of-date dependencies?" In *2017 32nd IEEE/ACM international conference on automated software engineering (ASE)*, IEEE, 2017, pp. 84–94.

[29] J. Musseau, J. S. Meyers, G. P. Sieniawski, C. A. Thompson, and D. German, "Is open source eating the world's software? measuring the proportion of open source in proprietary software using Java binaries," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 561–565.

[30] *National Vulnerability Database: Opaque changes and unanswered questions*, https://anchore.com/blog/national-vulnerability-database-opaque-changes-and-unanswered-questions/.

[31] S. Nocera, S. Vegas, G. Scanniello, and N. Juristo, "Software Composition Analysis and Supply Chain Security in Apache Projects: an Empirical Study," 2025.

[32] P. Ombredanne, "Free and open source software license compliance: Tools for software composition analysis," *Computer*, vol. 53, no. 10, pp. 105–109, 2020.

[33] H. Orsila, J. Geldenhuys, A. Ruokonen, and I. Hammouda, "Trust issues in open source software development," Jan. 2009.

[34] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vulnerable open source dependencies: Counting those that matter," in *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*, 2018, pp. 1–10.

[35] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vuln4real: A methodology for counting actually vulnerable dependencies," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1592–1609, 2020.

[36] I. Pashchenko, D.-L. Vu, and F. Massacci, "A qualitative study of dependency management and its security implications," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1513–1531.

[37] H. Plate, S. E. Ponta, and A. Sabetta, "Impact assessment for vulnerabilities in open-source software libraries," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2015, pp. 411–420.

[38] S. E. Ponta, H. Plate, and A. Sabetta, "Beyond metadata: Code-centric and usage-based analysis of known vulnerabilities in open-source software," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2018, pp. 449–460.

[39] S. Reis and R. Abreu, "A ground-truth dataset of real security patches," *arXiv preprint arXiv:2110.09635*, 2021.

[40] K. Scarfone and P. Mell, "An analysis of CVSS version 2 vulnerability scoring," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2009, pp. 516–525.

[41] P. Sharma, Z. Shi, Ş. Şimşek, D. Starobinski, and D. S. Medina, "Understanding Similarities and Differences Between Software Composition Analysis Tools," *IEEE Security & Privacy*, 2024.

[42] S. de Smale, R. van Dijk, X. Bouwman, J. van der Ham, and M. van Eeten, "No one drinks from the firehose: How organizations filter and prioritize vulnerability information," in *2023 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2023, pp. 1980–1996.

[43] J. Spring, E. Hatleback, A. Householder, A. Manion, and D. Shick, "Time to Change the CVSS?" *IEEE Security & Privacy*, vol. 19, no. 2, pp. 74–78, 2021.

[44] B. Stanton, M. F. Theofanos, S. S. Prettyman, and S. Furman, "Security fatigue," *It Professional*, vol. 18, no. 5, pp. 26–32, 2016.

[45] I. Steinmacher, T. Conte, M. A. Gerosa, and D. Redmiles, "Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '15, Vancouver, BC, Canada: Association for Computing Machinery, 2015, pp. 1379–1392.

[46] K.-J. Stol, M. A. Babar, P. Avgeriou, and B. Fitzgerald, "A comparative study of challenges in integrating open source software and inner source software," *Information and Software Technology*, vol. 53, no. 12, pp. 1319–1336, 2011.

[47] *The gift that keeps on giving: A new opportunistic Log4j campaign*, https://securitylabs.datadoghq.com/articles/the-gift-that-keeps-on-giving-a-new-opportunistic-log4j-campaign/, 2024.

[48] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, "How developers engage with static analysis tools in different contexts," *Empirical Software Engineering*, vol. 25, pp. 1419–1457, 2020.

[49] Y. Wang, B. Chen, K. Huang, B. Shi, C. Xu, X. Peng, Y. Wu, and Y. Liu, "An empirical study of usages, updates and risks of third-party libraries in java projects," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2020, pp. 35–45.

[50] D. Wermke, J. H. Klemmer, N. Wöhler, J. Schmüser, H. S. Ramulu, Y. Acar, and S. Fahl, ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain," in *44th IEEE Symposium on Security and Privacy (S&P'23)*, IEEE, May 2023, pp. 1545–1560.

[51] D. Wermke, N. Wöhler, J. H. Klemmer, M. Fourné, Y. Acar, and S. Fahl, "Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects," in *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*, IEEE, IEEE Computer Society, May 2022, pp. 1880–1896.

[52] Y. Wu, Z. Yu, M. Wen, Q. Li, D. Zou, and H. Jin, "Understanding the threats of upstream vulnerabilities to downstream projects in the maven ecosystem," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, IEEE, 2023, pp. 1046–1058.

[53] J. Wunder, A. Kurtz, C. Eichenmüller, F. Gassmann, and Z. Benenson, "Shedding light on CVSS scoring inconsistencies: A user-centric study on evaluating widespread security vulnerabilities," in *2024 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2024, pp. 1102–1121.

[54] A. Zerouali, T. Mens, A. Decan, and C. De Roover, "On the impact of security vulnerabilities in the npm and RubyGems dependency networks," *Empirical Software Engineering*, vol. 27, no. 5, p. 107, 2022.

[55] L. Zhao, S. Chen, Z. Xu, C. Liu, L. Zhang, J. Wu, J. Sun, and Y. Liu, "Software composition analysis for vulnerability detection: An empirical study on Java projects," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 960–972.

[56] S. Zhou, B. Vasilescu, and C. Kästner, "How has forking changed in the last 20 years? a study of hard forks on github," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 445–456.

[57] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem," in *28th USENIX Security symposium (USENIX security 19)*, 2019, pp. 995–1010.

A Interview Guide

S1 Participant Demographics
Can you tell us a little about the organization you are part of and your role?

S2 SCA Demographics
Main question: Can you briefly explain your experience with using SCA tools?

- What SCA tools are used in your workflow?
- What is the goal for using the SCA tool?
- How was the decision to pick this SCA tool made?
- If SCA tools are used in your organization: What kind of projects integrate SCA tools?
  - What projects do not integrate SCA tools?
  - What is the reason for not using SCA tools?
- Do you run SCA on source code or binaries?

S3 SCA tool process
Main question: Could you give us an example of how you use [SCA tool] in a project?

- How is [SCA tool] integrated into the SDLC?
- At which point of the SDLC is [SCA tool] triggered to run?
- How often are they triggered?
- What applications do you run [SCA tool] on? (web application, container images, directories, gh repos, microservices etc?)
- Is there a policy or standardized process at your organization that governs how you work with [SCA tool]?
- What are some challenges you have run into when running the tools?
- From your experience, do the tools perform better at certain programming languages or ecosystems?

S4 SCA tool report
Main question: Can you take us through a typical workflow for interpreting the output / report from [SCA tool]?

- What follows after receiving the output / report from [SCA tool]?
  - Is there an automation process for managing the SCA outputs?
- What do you look for when interpreting the output / report?
- How are warnings / alerts from the SCA tool resolved?
  - How is the decision on whether or not to resolve it made?
  - Who makes the call: developer / security engineer?
  - What is the relationship between developer / security engineer?
  - What factors affect the decision to resolve or accept risk?
- How are the warnings / alerts fixed?
  - Do you run into compatibility issues when updating dependencies?
- Is there a process for prioritizing the SCA alerts?

- What are some challenges when interpreting the output / report from [SCA tool]?
  - Have you run into FP / FN from [SCA tool]?
  - Can you walk us through how you would handle FP / FN?

S5 SCA tool features

- What are existing [SCA tool] features you think have been useful?
- What measures / signals / indicators from the tool have been useful?
- Do you make modifications/patches to [SCA tool]?
  - Can you describe how the modification/patch process works?
- What other features / measures do you wish were available in [SCA tool]?
  - Do you think reachability information would be useful?

S6 Participant opinion

- What is your opinion on SCA tools?
- What do you think about different SCA tools and the different results?
  - Was this considered when choosing SCA tools?
- What other information on vulnerabilities and CVEs do you wish are available?
- What do you think would help with SCA tools / vulnerable components going forward?
- What is an ideal SCA tool in your opinion? Which features do you value most?
- Anything relevant to the topic we haven't discussed but you want to mention?

B Codebook

1. Participant information
2. SCA tool usage
   (a) SCA tool used
   (b) Decisions on selection of SCA tool
   e.g., tool output, easy to setup
      i. switched tools
      e.g., easy to deploy, scan requirements
   (c) Goal of using tool
   e.g., basic security, compliance audits
3. SCA tool integration / process
   (a) Challenges / concerns
   e.g., code privacy, unknowns in how tool worked
      - SCA development
      e.g., package manager changes, identifying licenses
   (b) Integration
   e.g., ide plugin, CI/CD pipeline
      i. Type of application
      ii. SCA tool run frequency
      e.g., on code commit, every week
   (c) Policies
   (d) Modifications / patches to SCA tool
4. SCA tool report (output of the tool)
   (a) Postprocess output / automation tooling
   e.g., create ticket, resolve related vulnerability alerts
   (b) Resolving warnings / alerts
      i. Decision on whether to resolve
      ii. How to resolve
      e.g., updating dependencies, exception, isolate vulnerable component
      iii. Time to resolve
   (c) Prioritizing / interpreting alerts
   e.g., severity, project importance
   (d) FP / FN
      i. determining if alert is TP / FP
      e.g., manual review, developer decides
      ii. reasons for FP
      e.g., development dependencies, application not exposed
5. SCA tool features
   (a) Useful features
   e.g., automatic pull request
   (b) Less useful features / measures
6. Participant opinion
   (a) Better SCA tools / vulnerability management
   e.g., actionable findings, LLM inference
   (b) Different SCA tools and different results
   e.g., relevant results matter more, different results can be frustrating
   (c) Opinion on SCA tool
   e.g., false sense of security
   (d) Reachability
   (e) Vulnerability data
   e.g., CVEs need better standards