Military Cyber Affairs

Volume 7
Issue 1 *HammerCon 2024 Issue*

Article 6

May 2024

Surmounting Challenges in Aggregating Results from Static Analysis Tools

Dr. Ann Marie Reinhold
Montana State University, annmarie.reinhold@montana.edu

Brittany Boles
Montana State University, brittany.boles@student.montana.edu

A. Redempta Manzi Muneza
Montana State University, redempta.manzi@student.montana.edu

Thomas McElroy
Montana State University, thomasjmcelroy3@gmail.com

Dr. Clemente Izurieta
Montana State University, clemente.izurieta@montana.edu

Surmounting Challenges in Aggregating Results from Static Analysis Tools

Cover Page Footnote

This research was conducted with the U.S. Department of Homeland Security (DHS) Science and Technology Directorate (S&T) under contract 70RSAT22CB0000005. Any opinions contained herein are those of the authors and do not necessarily reflect those of DHS S&T. This is research is also supported in part by the Virtual Institutes for Cyber and Electromagnetic Spectrum Research and Employ (VICEROY) Northwest Virtual Institute for CyberSecurity Education & Research (CySER) program. The awarding agency is Air Force Office of Scientific Research (AFOSR) via Griffiss Institute. Sub Award contract 140178 SPC003260.

Surmounting Challenges in Aggregating Results from Static Analysis Tools

Dr. Ann Marie Reinhold, Brittany Boles, A. Redempta Manzi Muneza, Thomas McElroy, and Dr. Clemente Izurieta

Introduction

The abundance of analysis tools designed for detecting threats in software is so extensive that practitioners often find themselves overwhelmed when selecting tools for their respective organizations. The posture of an organization can be improved by applying the Platform for Investigative Software Quality Understanding and Evaluation (PIQUE) [12] framework, currently in development at the Software Engineering and Cybersecurity Laboratory (SECL) at Montana State University [16]. PIQUE assesses software artifacts from multiple sources, such as source code, binaries, Docker images, and Software Bills of Materials (SBOMs). Leveraging existing tools employed by an organization helps produce a holistic quality score to help improve assessments. While PIQUE is a framework that primarily addresses Quality Assurance (QA) concerns, we have developed parsimonious PIQUE models focused on cybersecurity.

The overarching purpose of our work is to bolster the cybersecurity defenses of organizations by moving beyond isolated point tools and, instead, incorporating the aggregation of assessments from multiple tools to inform practitioners with data from diverse sources. While the aggregation of measurements from diverse sources can enhance defenses, it also presents challenges.

The variability associated with aggregation from diverse sources is problematic. Such variability is often unconstrained and uncontrolled–leading to the propagation of inconsistencies and errors [8]. These errors and inconsistencies hamper accuracy and trustworthiness. Therefore, mitigating the effects of variability associated with aggregation is a research imperative as we engineer new approaches to bolster defenses in cyberspace.

Traditionally, the generation of reliable results has depended on the replication of studies. However, researcher attention needs to turn towards understanding the sources of variability (i.e., tools and technologies) and the internal algorithms (i.e., aggregation) used to process data produced from varied sources. It is crucial to acknowledge that the variability inherent in vendors, tool versions, third-party software, and host environments influences the outcomes of security assessments, with potential implications when selecting controls for an organization.

A formidable challenge arises when confronted with highly variable outputs stemming from different versions of the same tool. The complexity deepens in the face of disagreements between tools that measure similar aspects, numerous tool configurations, uncontrolled variability in dependencies, and diverse environmental conditions. We address this challenge by contributing our experience from rigorous assessments of multiple Static Analysis Tools (SATs). The variability of results reported by SATs is compounded when operationalizations of Quality Assurance (QA) models aggregate results to abstract quality characteristics (e.g., cybersecurity). Therefore, we advocate for a sound approach to aggregation to derive comprehensive and meaningful assessments which are subject to inputs from multiple sources.

Thus, making informed decisions about cybersecurity posture demands an acute awareness of tool variability, characterization of the dimensions of this variability, and a strategic approach to mitigate the impact of the aggregation of results that–in our specialized QA models–ultimately result in a security quality score. This research synthesizes the work we have carried out to date on these problems. Our research has implications for replicating and reproducing empirical studies, wherein consistency in their outcomes is crucial.

By sharing our diverse experiences with results obtained from several SATs and experimentation with aggregation techniques, we seek to enhance the reliability and trustworthiness of findings and, by extension, to improve the security assessments and posture of organizations. Beneficiaries of this research include Cybersecurity and Infrastructure Security Agency (CISA) Security Control Assessors (SCAs), Qualified Security Assessors (QSAs), Information Assurance Assessors (IAAs), and any personnel in similar work roles. To this end, we address the following goals (Figure 1):

G1: Report on the high variability of SATs.
G2: Report on a technique used to aggregate results from multiple sources.

The findings for each goal provide a deeper understanding of issues affecting variability stemming from SATs (G1) and aggregation algorithms (G2), thereby allowing our community to explore avenues for enhancing the operationalization of common canonical frameworks and improving the trustworthiness of security assessments.

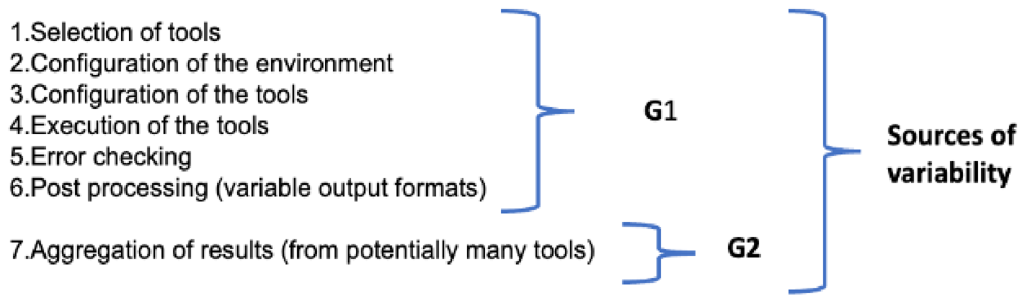

1.Selection of tools
2.Configuration of the environment
3.Configuration of the tools
4.Execution of the tools
5.Error checking
6.Post processing (variable output formats)
7.Aggregation of results (from potentially many tools)
G1
G2
Sources of variability

Figure 1: Sources of variability affecting practitioners who rely on aggregating the results of cybersecurity SATs. G1 and G2 refer to the goals of this paper; corresponding brackets indicate which sources of variation each goal addresses.

Related Work

Numerous tools are available for performing static analysis of source code, offering various strategies. Examples include SATs designed for assessing coding styles (e.g., Checkstyle [17]) and identifying source code warnings and bugs (e.g., Lint [2], PC-Lint [18], FindBugs [19], PMD [20], Roslynator [21], Security Code Scan [22], SonarQube [23], and Insider [24]). Additionally, specialized SATs focus on binary analysis to detect potential security threats, weaknesses, and vulnerabilities in programs. Examples include CVE Binary Tool [25] and CWE Checker [26].

While studies have compared functionalities across these SATs, there is a noticeable gap in research that examines a lack of consistency in the results reported by different versions of the same tool. Moreover, there is a lack of clear guidance regarding the vulnerabilities and weaknesses that these SATs target. For example, although two SATs may claim to report vulnerabilities, their outputs may vary significantly because of differences in the architecture, source code, or dependencies (e.g., databases).

Evidence of variability (from studies performed by members of SECL) in binaries and source code has been found and published in prior work. Reinhold et al. [11] conducted a study investigating the consistency with which multiple versions of cybersecurity SATs report weaknesses and vulnerabilities in binary files. They assessed CVE Binary Tool and CWE Checker. CVE Binary Tool reports vulnerabilities using Common Vulnerabilities and Exposures (CVE), whereas CWE Checker reports weaknesses using Common Weakness Enumeration (CWE). They conducted a systematic investigation of 660 unique binaries taken from a Kali Linux distribution, evaluated each binary with multiple versions of the SATs, and investigated how the findings changed according to the version of the static-analysis tool used.

Our ongoing research explores SATs designed for Docker containers and SBOMs. While this work is still in progress (see Table 1, cells indicated by the value "WiP"), preliminary findings suggest inconsistent reporting of results. Thus far, we have identified the following sources of variability: SAT vendors, different versions of SATs, dependencies on components and libraries, and output variations (Figure 1, Items 1-6). Further, we conjecture that many other sources of variability across tool domains exist but remain poorly investigated.

The challenges stemming from tool variation are further exacerbated when the evaluation techniques needed to combine scores from diverse sources require aggregation (Figure 1, Item 7), which can significantly threaten the internal and conclusion validity of security assessments. Aggregation refers to the combination of scores from multiple sources (e.g., tools), and is prominent in theoretical quality models that are organized in a hierarchical manner (e.g., International Standards Organization and the International Electrotechnical Commission, ISO/IEC 9126 [5] and ISO/IEC 25010 [6]). Quamoco [14] and the Quality Assessment Tool CHain (QATCH) [13] are examples of operationalizations of these models. Aggregation can pose significant problems for these operationalizations because the aggregation process involves assessing all measurements up the hierarchy using edge weights provided by the model. Aggregation methods range from simple calculations of weighted averages to more intricate linear transformations of data.

Tools created by different vendors produce different results, as exemplified in a technical debt study by Griffith et al. [3]. The authors empirically validated the relationships between technical debt scores produced by varying tools against established theoretical quality models. They conducted a case study across ten releases of ten open-source systems to evaluate three proposed methods of technical debt principal estimation. Their evaluation compared each technique against an external quality model. The authors found that only one estimation technique exhibited a strong correlation with the quality attributes of reusability and understandability.

Additional evidence of variability across tools and aggregation methods is found in a study conducted by Griffith et al. [7]. Therein, the authors employed a multiple case study design comparing the results of Quamoco [14] and SQALE [10] quality models; they used results from multiple tools (each from different vendors) as inputs to both Quamoco and SQALE. The study spanned various open-source C# software projects sourced from two sources: GitHub and commercial software for sustainment management systems. The assessments covered maintainability, reliability, and security. The findings indicated a notable disparity between the quality assessments under both models. This experiment showed variability across the quality models and across the tools used.

Variability in SAT results was also found by Rice [12] and Harrison [4]. Rice experimented with Rosylnator, a third-party tool suite that provides hundreds of code diagnostics. Rosylnator enables command line interface utilities to run static analysis on .NET systems. Rice also included the Security Code Scan tool for additional security measurements. Rice [12] documented variability in SAT results attributable to different vendors and their configurations. Additional work by

Harrison [4] expanded on work by Rice [12] using the Insider [24] security assessment tool within the PIQUE framework. Harrison [4] also demonstrates variability associated with configuration and tool vendors.

Uncertainty from multiple sources of variation propagates [8]. Such propagation has real consequences as margins of error expand. For instance, Brown et al. [1] further demonstrated the ripple effects linked to measurements associated with tool vendors.

Table 1: Cross reference of sources of variability against SATs in a specified domain. Values in cells indicate our experiences at the intersection of the sources of variability (rows) and the domain of the SATs (columns). "Published" denotes that we have published studies on the topics. "WiP" denotes work in progress, indicating that we have ongoing research. "Expected" denotes that the source of variability is relevant; relevance was determined using a combination of abductive reasoning that incorporates our previous experiences and expert knowledge. "Suspected" denotes that we suspect these sources of variability are important based on either abductive reasoning or expert knowledge.

Variability Source	*Binaries*	*Source Code*	*Docker Containers*	*SBOMs*
Version	*Published [11]*	*Expected*	*WiP*	*WiP*
Vendor	*Unexplored*	*Published [4], [7], [12]*	*WiP*	*Published [15]*
Configuration	*Expected*	*Published [4], [12]*	*Expected*	*Expected*
Failures	*WiP*	*Expected*	*WiP*	*WiP*
Outputs	*Expected*	*Expected*	*WiP*	*WiP*
Dependencies	*Published [9]*	*Expected*	*WiP*	*WiP*
Environment	*Suspect*	*Expected*	*Suspect*	*Suspect*

Methods

Over the last decade, our team has explored numerous open-source SATs that evaluate a range of software artifacts—from binaries, source code, Docker Containers, and SBOMs (Table 1). Our purposes for using these tools have varied widely, from investigating code quality to security to technical debt. All investigations have required aggregating results from multiple tools.

Our objective here is to exemplify and document the existence of variability when aggregating results from SATs (G1) and offer one solution for aggregating findings from multiple SATs (G2). Our analysis associated with G1 focuses on delineating the problem of reliance on one version of a SAT (e.g., the most recent version of the tool). Our method associated with Goal 2 offers an unbiased, tool-agnostic solution that we have developed to facilitate aggregating tool findings from multiple sources.

G1 Experimental Methods

To address G1, we initially outline (as detailed in Table 1) the potential sources of variability linked to each of the SATs we have investigated. It's important to clarify that the "Published" values in Table 1 pertain to work conducted by the authors and previous members of our SECL laboratory. It is beyond the scope of this article to provide summaries for all methods associated with our prior work. Here, we focus on experimentation done on *i)* binary analysis tools Common Vulnerabilities and Exposures (CVE) Binary Tool and Common Weaknesses Enumeration (CWE) Checker and *ii)* Docker Images analysis tools Grype [27] and Trivy [28]. These sources of variability address item 4 of G1 in Figure 1.

We present results that highlight two of the most common sources of variability that we have encountered: (1) variability arising from the use of different versions of a single SAT and (2) variability in the results of two SATs (from different vendors) that purportedly measure the same characteristic of a software artifact (Grype and Trivy). In both cases, the practitioner is left with unsettling, unquantified uncertainty–as they debate "*Which versions of this tool should I use?*" and "*Which vendor's results should I use?*" While we have experiences with additional sources of variability, we focus on version and vendor variability herein.

We highlight within-tool variability across tool versions employing two SATs that analyze binary code security [11] and two SATs that analyze Docker Images. We highlight between-tool variability by comparing the results of SATs that analyze Docker Images (WiP). We created a corpus of software artifacts to evaluate binaries and Docker Images. We evaluated one version of each binary file (n = 660) and one version of each Docker Image (n = 163), varying only the versions of the SATs that analyze the binaries and Docker Images. The same code artifacts were analyzed with multiple versions of multiple SATs.

Experimental method for comparing binary analysis SATs

We evaluated 660 publicly accessible binaries sourced from a Kali Linux distribution with multiple versions of CWE Checker and CVE Binary Tool [11]. The collection of binaries and the entire data-science workflow is downloadable from our GitHub [29] page, and detailed information is provided in [11]. For the CWE Checker, we evaluated all binaries using versions 0.4, 0.5, and 0.6. Earlier versions (0.1, 0.2, and 0.3) required deprecated dependencies, necessitating laborious and impractical modifications to the environment configuration. Consequently, our analysis concentrated on the recent versions (0.4, 0.5, and 0.6). We evaluated all binaries using nine versions of CVE Binary Tool. We attempted 11 versions, but omitted two versions because they appeared to have bugs; these two versions returned identical scores for all 660 binaries evaluated.

We implemented multiple controls to attribute variations in tool output to disparities in static analysis tool versions. First, we evaluated one version of each binary in the collection using multiple versions of the SATs. This ensured that all evaluated versions of the SATs analyzed the same binary code. Second, to ascertain that discrepancies in CVE Binary Tool output were solely attributable to variations in CVE Binary Tool versions and not influenced by differences in the National Vulnerability Database (NVD) [30] or the Open Source Vulnerabilities (OSV) database [31], we used NVD and OSV data acquired on July 18, 2022. Consequently, any differences observed in tool output across different CVE Binary Tool versions can be confidently attributed to the version of the tool, unaffected by variations in NVD or OSV versions. We calculated the total number of findings (CWEs from CWE Checker, CVEs from CVE Binary Tool) reported by each version of each tool. We assessed whether the version was an important driver of the variation in the results in Reinhold et al. [11].

Experimental method for comparing Docker container and SBOM analysis tools

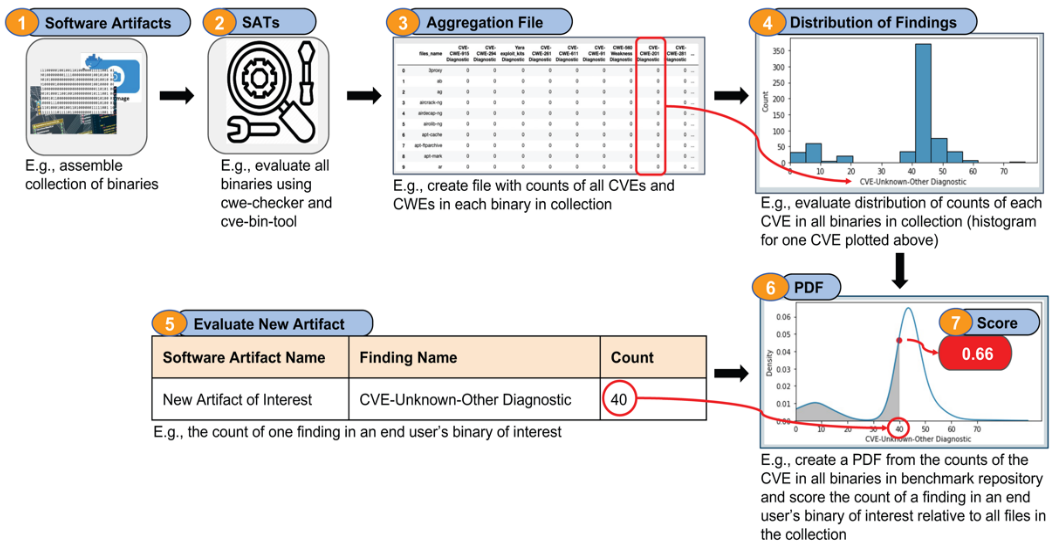
We evaluated a single version of each of the 163 Docker Official Images (i.e., containers) using the SATs Grype and Trivy. We collected these Official Images from Docker Hub and posted the version of the images on our public GitHub. We evaluated each Docker image with 22 versions of Trivy and 39 versions of Grype. The versions of each tool were major releases spanning from 12/24/2021 to 11/07/2023. In cases where a major release was unavailable, we used the next available sequential minor release. We omitted versions v0.38.0, v0.39.0, and v0.40.0 of Trivy from our analyses because executing each resulted in a runtime error. Each tool was run with default configurations with the following exceptions: (1) with Trivy, we used the configuration "-timeout 30m" to help minimize running errors, and (2) with Grype, we used the configuration "–scope -all layers" to ensure we did not have a configuration setting that resulted in false negatives (i.e., omitting vulnerabilities).

We implemented controls to attribute variations in tool output to disparities in static analysis tool versions. First, we evaluated one version of each Docker Image in the collection using multiple versions of the SATs. This ensured that all evaluated versions of the SATs analyzed the same set of Docker Images. Thus, each version of Grype and Trivy evaluated exactly one version of each Docker Image. We configured Grype and Trivy to use a static vulnerability database each; we posted links to each vulnerability database on our GitHub page. Grype and Trivy do not have common database formats, precluding us from using a single database for both SATs. However, each tool was executed with only one version of each database. Therefore, each tool analyzed each Docker Image using one vulnerability database, but the vulnerability databases were not identical between the two SATs.

We calculated the total number of vulnerabilities reported by each version of each tool. We present these graphically and provide basic descriptive statistics (e.g., standard deviation [SD]) to characterize our findings. We use the word "findings" here to represent the results produced by SATs.

G2 Experimental Methods

To address G2, we report on a procedure that we developed to aggregate results from diverse SATs. Different SATs provide results on different scales, precluding an end user from directly integrating findings from one with findings from another. Moreover, because no oracle exists to verify the findings of the SATs, ascertaining accuracy is a wicked problem. However, objective scaling of SAT findings is possible and enables aggregation of findings from diverse tools.


1 Software Artifacts
E.g., assemble collection of binaries
2 SATs
E.g., evaluate all binaries using cwe-checker and cve-bin-tool
3 Aggregation File
E.g., create file with counts of all CVEs and CWEs in each binary in collection
4 Distribution of Findings
CVE-Unknown-Other Diagnostic
E.g., evaluate distribution of counts of each CVE in all binaries in collection (histogram for one CVE plotted above)
5 Evaluate New Artifact
E.g., the count of one finding in an end user's binary of interest
6 PDF
7 Score
0.66
E.g., create a PDF from the counts of the CVE in all binaries in benchmark repository and score the count of a finding in an end user's binary of interest relative to all files in the collection

Software Artifact Name	Finding Name	Count
New Artifact of Interest	CVE-Unknown-Other Diagnostic	40

Figure 2: Density-based scoring procedure. Step 1: software artifacts are assembled into a collection. Step 2: all software artifacts in the collection are evaluated using a cadre of SATs. Step 3: the findings for all software artifacts evaluated by all SATs are aggregated. Step 4: the distribution of the count of findings is created. Step 5: an end user runs their new artifact of interest through the same SATs as in Step 3, and the instances of each finding are recorded. Step 6: a probability density function (PDF) is created from the distribution in Step 4. Step 7: the count of the finding in the new software artifact of interest receives a score between [0,1] based on its position in the PDF. All findings in the new artifact are scored with this density-based benchmarking procedure.

Our scaling method is rooted in comparison (Figure 2). We first create a collection of software artifacts (usually between $10^2 – 10^4$ artifacts). We evaluate all software artifacts with relevant static analysis tools (SATs); for instance, we use the SATs Trivy and Grype to analyze Docker Images and SBOMs. We then record the results for all findings reported by the SATs; typically, we store these in a flat file. Each file of this type is an "aggregation file."

In each aggregation file, the rows correspond to the software artifacts under investigation, the columns correspond to the findings (such as individual CVEs), and the values of each cell are the counts of each finding in each software artifact.

One file contains findings from multiple SATs; it could contain findings from multiple versions of multiple tools. However, our example here pertains to one file containing findings generated from one version of each SAT.

We then create a probability density function (PDF) based on the distribution of the instances of each finding in all samples in the collection. When an end user wishes to evaluate a new software artifact, they run it through the same SATs that the collection was evaluated with; each finding is then scored using a density-based approach. In the example in Figure 2, the end user's software artifact of interest is found to have 40 instances of the finding named "CVE-Unknown-Other-Diagnostic." The density-based scoring indicates that this artifact has fewer instances of this vulnerability than 66% of the artifacts in the collection.

To be more specific, suppose an end user has a software artifact that they are evaluating, and the count of a particular finding in that artifact is represented by d (we call this d because the count of a finding is "diagnostic"; d = 40 in Figure 2). We calculate a PDF for the distribution and compute the area under the curve from zero to d; thus, A_d indicates the proportion of samples in the collection having fewer counts of a finding than d. We calculate the score for the new artifact as $Score = 1 - A_d$; we subtract A_d from one because it is more intuitive for a higher score to indicate greater security and a lower score to indicate lower security.

Results

G1 Results

Our systematic evaluation of multiple versions of multiple SATs indicates that the findings from SATs are version-dependent. Multiple versions of the same tools produced different results in the collection of binaries (Figure 3 A-B and Reinhold et al. [11]) and in the collection of Docker Images (Figure 3 C-D). Thus, inter-version variation was present in the SATs that evaluate binaries and Docker Images.

Different versions of SATs often produce different measurements–even when the tool inputs (software artifacts) are identical. Our calculation of the standard deviation of the cumulative findings across the versions confirms this. While the magnitude of this variation is greater for the binary analysis SATs ($SD_{CVEBinaryTool} = 2.3e^4$, $SD_{CWEChecker} = 8.5e^4$), it is substantive for the SATs that analyze Docker Images also ($SD_{Trivy} = 4.2e^2$, $SD_{Grype} = 1.8e^3$). Note also that the scores for Trivy and Grype never agreed (the upper y-axis limit in Figure 3C [Trivy] is less than the lower bound of the y-axis limit in Figure 3D [Grype]).

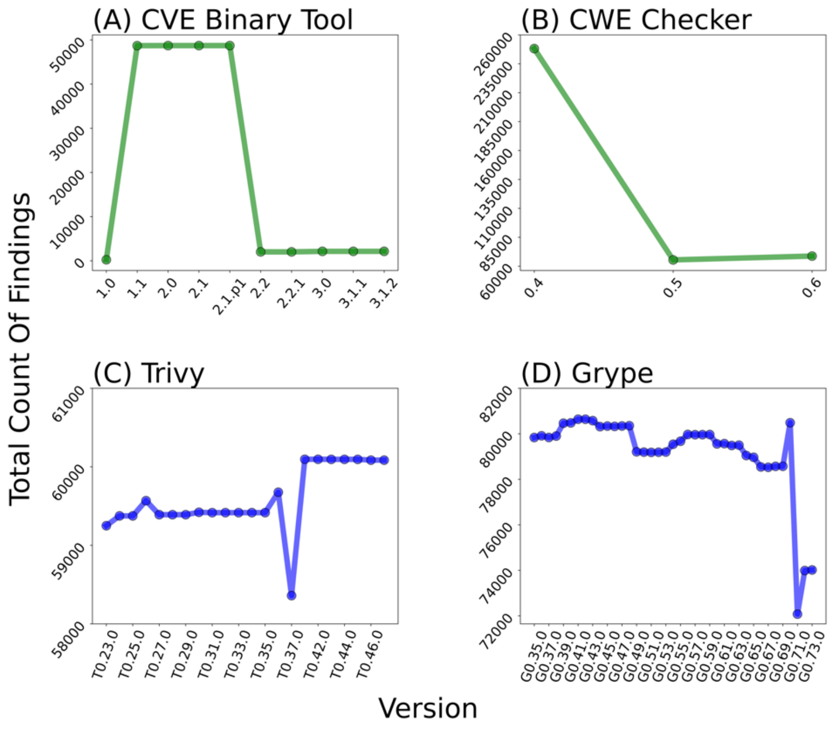


Figure 3: Total number of findings versus SAT. Exactly one version of each binary was analyzed by multiple versions of CVE Binary Tool and CWE Checker; exactly one version of each Docker Image was analyzed by multiple versions of Trivy and Grype. Y-axes correspond to the total count of findings identified by (A) CVE Binary Tool and (B) CWE Checker in the collection of binaries (n = 660) and by (C) Trivy and (D) Grype in the collection of Docker Images (n =163). X-axes correspond to the version of the SAT indicated in the panel labels. Each point represents the total number of findings that the SAT found in the collection of binaries (A-B) and Docker Images (C-D).

G2 Results

Our scoring procedure facilitates the aggregation of results from multiple sources with objectivity and transparency. We have successfully implemented this procedure to scale results from SATs that evaluate binaries, Docker Images, and SBOMs. In total, this amounts to over 13,000 software artifacts scored with this approach. Doing so has facilitated the aggregation of the SAT results into hierarchical QA models.

Our scaling procedure is robust across many distributions (Figure 4). In Figure 4, three distributions of findings from SATs and their associated PDFs are displayed. For two of these distributions, our procedure requires no modification; it produces intuitive and objective scores. However, our scoring procedure required one modification to accommodate the full suite of distributions that we have encountered. We expanded the density-based scoring procedure to accommodate the special case wherein all software artifacts in the collection had the same number of findings.

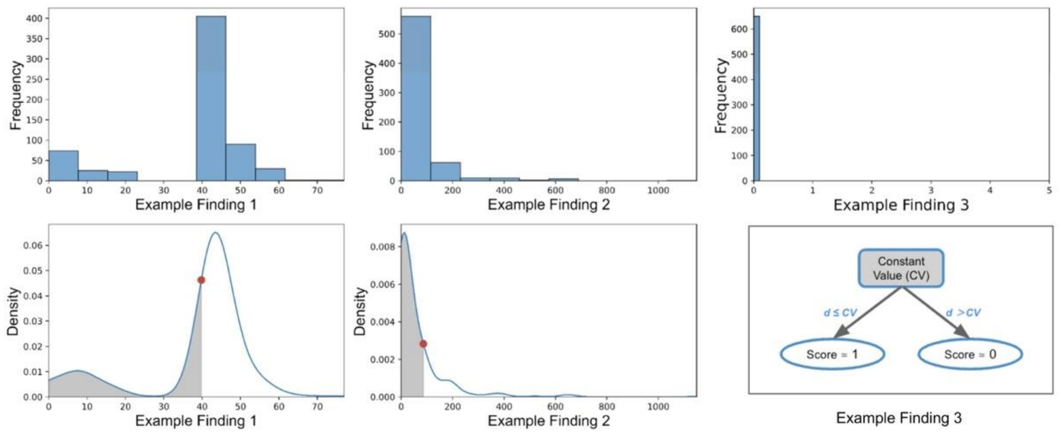


Figure 4: Density-based scoring of commonly encountered distributions. The upper row depicts histograms of distributions that we observed in benchmarking our collections of software artifacts. The lower row depicts how application of the density-based scoring to produce a score between [0,1]. Red dots denote the count of findings in a software artifact compared to all other artifacts in a collection. Note that no PDF is plotted in the lower right panel. A PDF is nonsensical in the case where the findings for all software artifacts in the collection were identical (here, all were zero); we refer to these as "constant values." When encountering constant values in a benchmark repository, the approach is to assign a score of zero if d is less than or equal to the constant value and to assign a score of one if d is greater than the constant value.

Discussion

Our results from G1 illustrate the magnitude of the variation in the number of findings reported by SATs while changing only the version of the SAT itself. Developer change logs sometimes help explain the large changes in findings reported by the SATs. For instance, the total number of findings identified by Grype dropped substantively following v0.70.0 (Figure 3D). This drop in findings is likely attributable to the developers moving away from using Common Platform Enumeration (CPE) matching in subsequent versions, thereby changing how Grype interacts with its dependencies (e.g., how pattern matching is done with the NVD). This is an example where change logs explain variation, but this level of explanation is rare in our experience. Often, we are challenged to ascertain the reasons for the inter-version variation.

The inter-version variation presents a formidable challenge for selecting the appropriate version of a SAT. Moreover, high inter-version variation hampers end-user trust in the accuracy of the findings reported by a SAT. Therefore, inter-version variation has become an important metric for informing which SATs we use in our security assessment studies because any errors or inconsistencies introduced by the SATs propagate in our models and subsequent analyses.

When two SATs purportedly measure the same (or very similar) aspects of a software artifact, inter-version variation can help inform SAT selection. Selection criteria can and should still include a comprehensive evaluation of developer documentation and full testing of the tools. We have a study underway that investigates the technical reasons for the differences in the results that Grype and Trivy produce when evaluating Docker Images and SBOMs. Selection criteria should also incorporate which SATs produce consistent results across versions and the extent to which change logs and release notes explain large changes in tool findings. The findings reported by the tools will change commensurate with the software development lifecycle; however, large, unexplainable changes in findings suggest that the SATs are not yet mature enough to be trusted.

The solution we present in G2 is appealing because it is unbiased, repeatable, and effective at normalizing results from diverse sources (here, multiple SATs) to enable aggregation. However, it has some limitations. The external validity and accuracy of inference derived from this solution are heavily dependent on the representativeness of the collection of artifacts against which a new artifact is compared. For example, suppose a software engineer was interested in analyzing the security of compiled ladder logic code written for a Programmable Logic Controller (PLC) using the same collection of binaries analyzed in Reinhold et al. [11] and here (Figure 3 A-B). This collection of binaries was sourced from a Kali Linux distribution, so scoring the compiled PLC code against this collection would introduce threats to validity. More colloquially, this would be an “apples to oranges” comparison because our collection of binaries contains no compiled code samples from PLCs. However, the threats to validity are much smaller if, for instance, a software engineer wanted to compare findings from a binary from another Debian-based Linux distribution against our collection of binaries; this would be more of an “apples to apples” comparison.

The scaling approach presented here can be adapted to facilitate aggregation for other SATs and in other contexts but may require modification in certain use cases. We developed this procedure to enable aggregation of cybersecurity findings from SATs. For this use case, inferring higher security and lower security from SAT results is clear. Counts of vulnerabilities and weaknesses are inversely related to the security of software artifacts. That is, high counts of vulnerabilities and weaknesses indicate poor security whereas low counts indicate high security.

We have not yet expanded our scaling procedure to aggregate results from SATs that measure aspects of code quality wherein “good" values are intermediate (neither high nor low). Coupling is one such aspect. A “good” metric for coupling is neither zero nor one. Rather “good” coupling values tend to be somewhere between these values, with variation being dependent on the domain of a software artifact. We have only just begun adapting our density based scoring to attributes such as coupling. This is a subject for our future work.

Lessons Learned

While we only addressed version and vendor as sources of variability in the analyses in this paper, we identified at least 7 sources (Table 1). Assessing these sources of variability simultaneously as a whole is an unwieldy task. We assert that breaking each component down into more atomic components will facilitate understanding the nuances of each source of variability. Moreover, we posit that there is value in applying a familiar organizational structure as we unpack the effect that each source of variability has on the wicked problem of aggregating results.

Here, we apply first principles from quality modeling (e.g., ISO/IEC 25010 [6]) to organize these sources of variability (Table 1). While many of these sources of variability touch on multiple quality characteristics (i.e., a many-to-many relationship exists between sources and quality aspects), we offer a primary classification for these sources of variability as a first step towards developing a taxonomy. The variability sources "vendor" and "environment" map to the concept of portability defined by the ISO standard. "Configuration" and "dependencies" fall predominately under the usability characteristic. "Version" pairs with the compatibility characteristic; "failures" pairs the reliability characteristic, and "dependencies" pairs with the functional suitability characteristic.

Future Directions

The solution that we present here (G2) is applicable for aggregating information across a range of sources where no oracles exist. Still, we have yet to scratch the surface of delineating and solving the problem of aggregation in security assessments and empirical software engineering in general. The empirical software engineering community can benefit from researching and adopting a common set of organizing principles, metrics, and standards to guide the aggregation of data from disparate sources. Adopted principles from this community will trickle down and enhance assessments for practitioners—including those conducting cybersecurity assessments.

Inconsistencies and errors propagate during aggregation, threatening the validity of results and any inferences drawn from them. Further, post hoc identification and correction of inconsistencies to facilitate aggregation often leads to uncertainty [8]. Our SECL team's future work is focused on better delineating the problems with aggregation and how uncertainty and inconsistency in sources impact inference. To this end, we are preparing a systematic literature review to understand the extent to which our experiences with aggregating the findings of SATs are representative of the empirical software engineering community. We seek to create a taxonomy of these sources of variability and build towards a common set of "best practices" for aggregation. This work is important to ensuring that PIQUE models are high quality and will benefit a broad community of practitioners reliant on aggregation.

Conclusions and Recommendations

Sources of variability compound uncertainty. When assessing the security posture of organizations, the compounding of uncertainty is a pressing problem. Our results indicate that our community would benefit from better characterizing and quantifying this uncertainty and its sources. Here, we demonstrate that version and vendor are common sources of variability in studies aggregating findings from SATs.

Aggregation remains a wicked problem because the combinatorics of the many sources of variation are vast. Selecting a vendor and a version are universal choices that software engineers are confronted with across myriad use cases, but many more sources of variation exist. We highlight several of these sources of variation and organize them to provide bounds on this challenge. We recommend solutions to aggregation that are organized around common goals and are broadly applicable across use cases.

About the Authors

Dr. Ann Marie Reinhold

Dr. Ann Marie Reinhold is an Assistant Professor in the Gianforte School of Computing and Co-Director of the Software Engineering Lab at Montana State University. She specializes in the development and application of computational methods to understand the mechanisms underpinning pressing environmental, societal, and cybersecurity problems. https://www.linkedin.com/in/amreinhold/

Brittany Boles

Brittany Boles is a first-year master's student at Montana State University studying cybersecurity. She received a bachelor's degree in applied mathematics from Montana State University. She worked as a Software Engineer for S2-Corporation, a photonics company centered around defense, for two years. In her research, she takes a data science approach to understanding the inner workings of cybersecurity static analysis tools. https://www.linkedin.com/in/brittany-boles-7026202a3

A. Redempta Manzi Muneza

Redempta is a Ph.D. student in the Gianforte School of Computing at Montana State University. Her current work focuses on the implementation of data science approaches to improve the analysis of cybersecurity threats in software systems. Her research interests are software security, security risk assessment, and machine learning. She holds a master’s degree in computer science from Boise State University. https://www.linkedin.com/in/manzi-muneza-assoumer-redempta/

Thomas McElroy

Tom McElroy is currently pursuing a Ph.D. in Computer Science at Montana State University. He holds a B.S. in Physics from Florida State University and a M.S. degree in Computer Science from the University of West Florida. He worked as an electronic warfare engineer at Eglin Air Force Base for 5 years. His current research interest is the application of data science and machine learning techniques to remote sensing.

Dr. Clemente Izurieta

Dr. Izurieta is a Professor in the Gianforte School of Computing at Montana State University and Co-Director of the Software Engineering and Cybersecurity Laboratory. He obtained his Ph.D. from Colorado State University, and his research interests include empirical software engineering, design and architecture of software systems, design patterns, and the measurement of software quality and cybersecurity. https://www.linkedin.com/in/cizurieta/

References

[1] Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, Raghvinder Sangwan, Carolyn Seaman, Kevin Sullivan, and Nico Zazworka. 2010. "Managing technical debt in software-reliant systems." In Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (Santa Fe, New Mexico, USA) (FoSER '10). Association for Computing Machinery, New York, NY, USA, 47–52. https://doi.org/10.1145/1882362.1882373

[2] Ian F. Darwin. 1996. "Checking C programs with Lint." O'Reilly.

[3] Isaac Griffith, Derek Reimanis, Clemente Izurieta, Zadia Codabux, Ajay Deo, and Byron Williams. 2014. "The correspondence between software quality models and technical debt estimation approaches." 2014 Sixth International Workshop on Managing Technical Debt (2014). https://doi.org/10.1109/mtd.2014.13

[4] Payton Harrison. 2022. "Analyzing the security of C# source code using a hierarchical quality model." Master's Thesis. Montana State University.

[5] ISO. 2012. ISO/IEC 9126-1:2001 Software Engineering Product Quality Part 1: Quality Model. Retrieved November 28, 2023, from https://www.iso.org/standard/22749.html

[6] ISO. 2017. ISO/IEC 25010:2011 Systems and Software Engineering Systems and Software Quality Requirements and Evaluation (Square) System and Software Quality Models. Retrieved November 28, 2023, from https://www.iso.org/standard/35733.html

[7] Clemente Izurieta, Isaac Griffith, and Chris Huvaere. 2017. "An industry perspective to comparing the SQALE and Quamoco software quality models." In 2017, ACM/IEEE International Symposium on Empirical Software

Engineering and Measurement (ESEM). 287–296. https://doi.org/10.1109/ESEM.2017.42
[8] Clemente Izurieta, Isaac Griffith, Derek Reimanis, and Rachael Luhr. 2013. “On the uncertainty of technical debt measurements.” 2013 International Conference on Information Science and Applications (ICISA) (2013). https://doi.org/10.1109/icisa.2013.6579461
[9] Andrew Lucas Johnson. 2022. “The analysis of binary file security using a hierarchical quality model.” Ph.D. Dissertation. Montana State University. https://scholarworks.montana.edu/xmlui/handle/1/16635
[10] Jean-Louis Letouzey and Thierry Coq. 2010. “The SQALE Analysis Model: An analysis model compliant with the representation condition for assessing the quality of software source code.” 2010 Second International Conference on Advances in System Testing and Validation Lifecycle (2010). https://doi.org/10.1109/valid.2010.31
[11] Ann Marie Reinhold, Travis Weber, Colleen Lemak, Derek Reimanis, and Clemente Izurieta. 2023. “New version, new answer: Investigating cybersecurity static-analysis tool findings.” 2023 IEEE International Conference on Cyber Security and Resilience (CSR) (2023). https://doi.org/10.1109/csr57506.2023.10224930
[12] David Rice. 2020. “An extensible, hierarchical architecture for analysis of software quality assurance.” Master’s Thesis. Montana State University.
[13] Miltiadis G. Siavvas, Kyriakos C. Chatzidimitriou, and Andreas L. Symeonidis. 2017. “Qatch - an adaptive framework for software product quality assessment.” Expert Systems with Applications 86 (2017), 350–366. https://doi.org/10.1016/j.eswa.2017.05.060
[14] Stefan Wagner, Klaus Lochmann, Sebastian Winter, Florian Deissenboeck, Elmar Juergens, Markus Herrmannsdoerfer, and Lars Heinemann. 2012. “The Quamoco quality meta-model.” Technical Report.
[15] Eric O’Donoghue, Ann Marie Reinhold, Clemente Izurieta. 2024. “Assessing security risks of software supply chains using software bill of materials.” 2nd International Workshop on Mining Software Repositories for Privacy and Security (MSR4P&S).
[16] Software Engineering and Cybersecurity Laboratory (SECL). Available online at: https://www.montana.edu/cyber
[17] Checkstyle. Available online at https://checkstyle.sourceforge.io/
[18] Pclintplus. Available online at https://pclintplus.com/
[19] Findbugs. Available online at https://findbugs.sourceforge.net/
[20] Pmd. Available online at https://pmd.github.io/
[21] Roslynator. Available online at https://github.com/dotnet/roslynator
[22] Security Code Scan. Available online at https://security-code-scan.github.io/
[23] SonarSource. Available online at https://www.sonarsource.com/products/sonarqube/
[24] Insidersec. Available online at https://github.com/insidersec/insider
[25] CVE bin Tool. Available online at https://github.com/intel/cve-bin-tool
[26] CWE Checker. Available online at https://github.com/fkie-cad/cwe_checker
[27] Grype. Available online at https://github.com/anchore/grype

[28] Trivy. Available online at https://github.com/aquasecurity/trivy
[29] MSU SECL Static Analysis Comparison. Available online at https://github.com/MSUSEL/MSUSEL_SAT_Comparison
[30] National Vulnerability Database. Available online at https://nvd.nist.gov/
[31] Open Source Vulnerabilities. Available online at https://osv.dev/