Semantic Similarity-Based Clustering of Findings From Security Testing Tools

Phillip Schneider[1*], **Markus Voggenreiter**[2*], **Abdullah Gulraiz**[1*] **and Florian Matthes**[1]
[1]Technical University of Munich, Department of Computer Science, Germany
[2]Siemens Technology & LMU Munich, Germany
{phillip.schneider, abdullah.gulraiz, matthes}@tum.de
markus.voggenreiter@siemens.com

Abstract

Over the last years, software development in domains with high security demands transitioned from traditional methodologies to uniting modern approaches from software development and operations (DevOps). Key principles of DevOps gained more importance and are now applied to security aspects of software development, resulting in the automation of security-enhancing activities. In particular, it is common practice to use automated security testing tools that generate reports after inspecting a software artifact from multiple perspectives. However, this raises the challenge of generating duplicate security findings. To identify these duplicate findings manually, a security expert has to invest resources like time, effort, and knowledge. A partial automation of this process could reduce the analysis effort, encourage DevOps principles, and diminish the chance of human error. In this study, we investigated the potential of applying Natural Language Processing for clustering semantically similar security findings to support the identification of problem-specific duplicate findings. Towards this goal, we developed a web application for annotating and assessing security testing tool reports and published a human-annotated corpus of clustered security findings. In addition, we performed a comparison of different semantic similarity techniques for automatically grouping security findings. Finally, we assess the resulting clusters using both quantitative and qualitative evaluation methods.

1 Introduction

The automation of security tests is a common practice for software engineering projects that apply software development and operations (DevOps) practices. Different security tools employ different perspectives to scan a software artifact as part of Continuous Integration or Continuous Deployment (CI/CD) pipelines, producing semi-structured reports of security findings. While this approach fosters DevOps principles, reduces manual effort, and shifts security efforts to the earlier stages of development, it also comes at a cost.

Since security testing tools often have an overlapping scanning coverage, duplicates or nearly identical findings are unavoidable. Further, considering that each iteration brings new security findings, identifying duplicate security findings is essential to achieve a reliable overview. In this context, it is important to note that we define *duplicates* as findings that point out the exact same security problem, potentially occurring at multiple locations in the software. Exemplary for that would be an SQL injection vulnerability at multiple locations of a web interface. Amongst multiple other activities, the identification of duplicates is traditionally addressed by a team member with security domain knowledge, a so-called security professional, before looping back the security findings to development to improve the software security-wise (Simpson, 2014). Taking the frequency of new reports and the number of findings throughout all security tests into account, an entirely manual analysis is unfeasible, prone to human error, and violates fundamental DevOps principles.

Natural Language Processing (NLP) has been shown to be effective in analyzing and clustering textual data from various application domains, such as medicine, linguistics, and software engineering (Demner-Fushman and Lin, 2006; Majewska et al., 2018; Aggarwal et al., 2017). Although security tool reports contain highly domain-specific text, it seems promising to investigate NLP techniques for automatically grouping findings into problem-oriented clusters, which can assist security professionals in their analyses. To our best knowledge, no studies specifically focus on the machine-generated finding texts produced by security scanning tools.

* The first three authors have contributed equally.

Addressing this research gap, we evaluated the performance of three common semantic similarity techniques. The selected techniques originate from knowledge-based, corpus-based, and neural network-based methods. Our main contributions are twofold:

1. We publish a human-annotated corpus of clustered security findings along with the annotation tool used by the security professionals.

2. We perform an in-depth analysis of three popular semantic similarity techniques for clustering security findings, followed by a quantitative and qualitative evaluation of the results.

The remainder of this paper is structured as follows. Section 2 presents background information on security scanning tools and gives an overview of related work on applying NLP techniques in the software engineering domain. Section 3 describes the employed two-stage research approach for the dataset construction and experimental evaluation. We report the clustering results, discuss our observations, and outline the limitations in Section 4, Section 5, and Section 6, respectively. Section 7 concludes the paper with a summary and an outlook toward future work.

2 Background and Related Work

This section provides background information on security testing tools and security finding reports in DevOps. Furthermore, we mention related studies concerning the application of NLP techniques in the software engineering domain.

To tackle the challenge of duplicates in security reports, we first establish the definition of what duplicate security findings are. We consider two findings to be duplicates if they describe the exact same problem at any location of the software. Consequently, the same issue, e.g., an SQL injection, could occur at multiple places but would be considered a duplicate. Besides the problem-based approach, other strategies for describing duplicates can also incorporate the location of a finding or its underlying solution. The selection of a strategy in this area highly depends on the subsequent actions on the dataset.

Furthermore, it is necessary to explain the activities that generate security reports that contain duplicate findings. Security testing can be categorized according to multiple properties depending on the testing strategy, involved testers, tested components, and numerous others. We limit our categorization to those security tests that can be automated in pipelines and scan an actual part of the product. Further, we categorize them into two major categories: tests that examine the static elements of the software (e.g., code, configuration, or dependencies) are called static application security testing (SAST) and tests performed against the dynamic, actually running application are called dynamic application security testing (DAST). This separation represents a clear distinction, as static testing can only guess whether a finding is actually affecting the software, while dynamic techniques directly identify the exploitable security finding.

From our analysis of the literature on security findings management, we found that there are no NLP-related publications that focus on the identification of duplicate security findings. However, a number of NLP methods have been successfully applied to related subdomains in the software engineering field. For example, Kuhn et al. (2007) use latent semantic indexing (*LSI*) and clustering to analyze linguistic information found in source code, such as identifier names or comments, to reveal topics and support program comprehension. In a study from Schneider (2020), a corpus of app reviews with comments about a variety of software issues is clustered into topics with problem-specific issue categories. Another study from Eyal Salman et al. (2018) focuses on automatically forming semantic clusters of functional requirements based on cosine similarity with a corpus of documents containing software requirements specifications. The authors conduct an empirical evaluation of agglomerative hierarchical clustering using four open-access software projects. In order to assess the software quality of programs, Tan et al. (2011) apply a hierarchical cluster algorithm to create problem-oriented clusters, reducing the effort needed to review the code. The study shows that semantic clusters are an effective technique for defect prediction.

3 Method

In order to achieve our objective of investigating semantic similarity techniques for clustering findings from security testing reports, we constructed a human-annotated dataset. This annotated corpus consists of 1351 SAST and 36 DAST findings. The two-stage process with dataset construction as well as experimental evaluation is explained in the following subsections.

3.1 Dataset Construction

To quantify the performance of different semantic similarity techniques, a ground-truth benchmark dataset is required, enabling the comparison between human-labeled clusters and the predictions of the semantic similarity algorithms. Therefore, we asked two security professionals from the industry to annotate semantically duplicate findings in a given list of security reports. Due to the significant differences in perspective between SAST and DAST reports, we decided to construct two separate datasets, each of which comprising reports from only one testing type.

A major challenge in constructing such a dataset is the content of the security tool reports. Security tool reports are often exported as JSON files containing security finding objects. Across different tools, these reports utilize different schemas, resulting in different property names referring to the same finding feature (e.g., *description*, *FullDescription*, *text*, *Message*, or *details*). For the construction, the security professionals consolidate semantically duplicate findings from all tool reports of a testing iteration based on certain features, e.g., description, location, or unique identifier. Therefore, they need to find the feature in the respective tool schema and compare it to the other findings. Manually annotating such a dataset would require them to memorize $N \times M$ property names when identifying N features across M distinct security testing reports. To enhance efficiency and reduce manual, repetitive work, we developed the Security Findings Labeler (*SeFiLa*).[1] This tool allows security professionals to upload reports from different security tools and conveniently group all findings into named clusters.

The initial, unconsolidated reports of the dataset were generated by scanning the open-source, vulnerable web application JuiceShop[2] with seven SAST tools and two DAST tools. For reproducibility reasons, we solely selected tools free of charge that can be reasonably automated in real-world software development pipelines. We selected Anchore, Dependency Check, Trivy, HorusSec, Semgrep, CodeQL, and Gitleaks as SAST tools. For DAST, we selected Arachni and OWASP ZAP. Fundamental information about each tool can be found in Table 5 in the appendix. From each tool, one testing report was taken for the dataset. The security professionals assigned findings to named clusters representing the same security problem. This process was aided by features like the CVE-ID (common vulnerabilities and exposures) which provides an identifier and a reference-method for publicly known security vulnerabilities. Other helpful features are descriptions and solutions generated by the testing tools. After all findings were assigned to clusters, the dataset comprising our baseline for duplicate identification was completed. The dataset and the code to run the test cases were published in a public GitHub repository.[3]

3.2 Evaluation Procedure

For conducting the evaluation, we investigated semantic similarity methods proposed in the literature and chose three popular techniques that are often used as baseline models: knowledge graph-based similarity with *WordNet* (Miller, 1995), *LSI* (Landauer and Dumais, 1997), and *SBERT* (Reimers and Gurevych, 2019). To evaluate the semantic similarity techniques, we extracted all findings from the security testing tool reports and concatenated selected features from them to form problem-specific finding strings. We applied the three chosen semantic similarity techniques to the finding strings to determine those that are semantically similar. Since semantic similarity between two finding strings is calculated as a score between 0 and 1 where 1 indicates highest similarity, we established a *similarity threshold* for each experiment. This threshold defines the value above which two finding strings are deemed to be semantically similar. Findings corresponding to these similar finding strings are then grouped to form predicted clusters. Implementation-wise, predicted and ground-truth clusters both consist of unique integer sequences, each integer representing a finding from the dataset.

Before the clusters were compared with each other in the quantitative evaluation, we encountered the need for *transitive clustering* of findings. In certain cases, the problem description of two findings was identical, but it was repeated in one finding for multiple instances, leading to a discrepancy in text length. Since the similarity depends on the similarity of the finding strings, we encounter the following example predictions with *Similar Findings* listed in descending order of semantic similarity scores with the corresponding *Finding* identifier:

[1] https://github.com/abdullahgulraiz/SeFiLa

[2] https://owasp.org/www-project-juice-shop/

[3] https://github.com/abdullahgulraiz/SeFiDeF

$$\{Finding: 1,\ Similar\ Findings: \{1, 2, 4\}\}$$

$$\{Finding: 2,\ Similar\ Findings: \{2, 1, 3, 5\}\}$$

Let us assume that findings $\{1, 2, 3\}$ contain the same problem description, although it appears once in $Finding$ 1, two times in $Finding$ 2, and three times in $Finding$ 3. While $Finding$ 1 is found similar to findings $\{1, 2, 4\}$, its similarity score with respect to $Finding$ 3 is below the clustering threshold due to the different text length. However, $Finding$ 2 does have $Finding$ 3 in its set of similar findings. If $Finding$ 3 is similar to $Finding$ 2, it should also be similar to $Finding$ 1, regardless of repetitive text. Therefore, even though $Finding$ 3 exists only in the set of similar findings for $Finding$ 2, it should appear in the final set of similar findings of $Finding$ 1 as well. In our initial clustering experiments and discussions with the security professional, we observed that while lowering the similarity threshold led to many false positive predictions, transitive clustering improved the results without changing the similarity threshold. Therefore, we apply the transitive property to consider findings as semantically related through *intermediate* findings. This causes the above predictions to become:

$$\{Finding: 1, SimilarFindings: \{1, 2, 3, 4, 5\}\}$$

$$\{Finding: 2, SimilarFindings: \{1, 2, 3, 4, 5\}\}$$

After transitive clustering, we removed the duplicate clusters from predictions and evaluated the final predictions against the ground-truth clusters.

Table 1 shows a contingency matrix that illustrates possible outcomes when comparing clusters from predictions (P) with clusters from the ground-truth dataset (Q). The number of occurrences of these outcomes is used to calculate the metrics of *precision*, *recall*, and *F-score*.

	Predictions (P)			
	Clusters in P		Clusters not in P	
Ground-truth (Q)				
Clusters in Q	True Positive (TP)		False Negative (FN)	
Clusters not in Q	False Positive (FP)		True Negative (TN)	

Table 1: Contingency matrix of predicted clusters P and ground-truth clusters Q.

The *precision* (Hossin and Sulaiman, 2015) measures positive patterns correctly predicted from the total predicted patterns in a positive class. In our experiments, it measures the ratio of correct cluster predictions to all predictions. Higher precision indicates that less false positive predictions appeared in the results. It is calculated as:

$$Precision = \frac{TP}{TP + FP}$$

The *recall* (Hossin and Sulaiman, 2015) is used to measure the fraction of correctly classified positive patterns. In our experiments, it represents the ratio of correctly predicted clusters to all ground-truth clusters. A high recall value thus indicates that the semantic clustering results retrieve many ground-truth clusters of the security professional.

$$Recall = \frac{TP}{TP + FN}$$

The *F-score*, also known as Dice Measure (Dice, 1945), calculates the harmonic mean between precision and recall. It balances both metrics to provide an overall performance overview.

$$F - score = \frac{2 * TP}{2 * TP + FP + FN}$$

In addition to the quantitative evaluation with performance measures, we collected qualitative feedback from the security professionals on incorrectly clustered findings. We limited the information about each finding to the finding strings used as input for the NLP techniques and asked for possible reasons for the incorrect clustering. This created a list of reasons that led to poor duplicate identification from the perspective of a domain-aware security professional. Finally, each incorrect cluster is associated with at least one reason for the incorrect clustering, providing insights into the different challenges and their prevalence in the results. The evaluation was aided by *SeFiLa* for annotation of the findings, assignment of reasons, and documentation.

4 Experiments

4.1 Dataset Description

After labeling the exported security findings with our annotation tool *SeFiLa*, the security professionals provided us with two datasets, namely the manually grouped SAST and DAST findings. The descriptive statistics of both datasets are summarized in Table 2. We observe that SAST findings

Statistic	SAST	DAST
Number of clusters	183	10
Number of findings	1351	36
Avg. findings per cluster	7	3
Avg. characters per finding	302	471
Min. findings per cluster	1	1
Max. findings per cluster	408	25

Table 2: Data records from static analysis security tools (SAST) and dynamic analysis security tools (DAST).

are far more frequent, making up 97.4% of the total findings. The number of formed clusters for the SAST findings is significantly higher than for DAST findings. While both datasets had clusters with only one finding, the maximum cluster size was by far larger in the SAST dataset. Despite these discrepancies, the average number of findings per cluster is not too different between the datasets, ranging from a mean value of 3 for DAST to a mean value of 7 for SAST findings. In addition, DAST finding texts are more verbose since they contain 169 more characters on average. To investigate the potential of semantic similarity techniques, constructing the finding string from the finding features is crucial. Analyzing the initial dataset, we identified that solely a single feature describing the finding is consistently found across all SAST tools. For the DAST findings, multiple features, including the description, a name, and even a solution/mitigation, were consistently found across all findings. Furthermore, we observed that DAST features are sufficiently verbose to comprehend the problem from their finding string and thereby contain enough semantic content for NLP. Contrarily, we find SAST features to be very brief, for that matter, making it almost impossible to understand a finding just from the finding string.

To counteract the limitation of very short SAST finding strings, we make use of CVE-IDs to increase the textual content of SAST finding strings. By leveraging the CVE identifier present in some findings, we concatenated finding strings of various machine-generated descriptions with the same CVE-ID. This allows for more semantic content and longer descriptions about the underlying problem. We used the concatenated finding strings as input to the NLP-based similarity techniques.

This step led us to construct a total of four corpora with finding strings from both SAST and DAST datasets for the identification of duplicate findings, as listed below:

- ***SAST-D*:** consists only of SAST finding descriptions
- ***SAST-ConcD*:** consists of concatenated SAST finding descriptions with the same CVE-ID
- ***DAST-NDS*:** consists of concatenated DAST finding names, descriptions, and solution texts
- ***DAST-D*:** consists only of DAST finding descriptions

4.2 Evaluation Results

The summary of the quantitative results achieved when applying semantic clustering using a technique from each category of semantic similarity methods to each of the four corpora is presented in Table 3. The experiments were performed for similarity thresholds $0.1 \leq$ and ≤ 0.95. The performance metric values for the experiment with the highest F-score are reported.

Technique	Corpus	Metrics		
		F-score	Precision	Recall
SBERT	*SAST-D*	0.709	0.621	0.825
	SAST-ConcD	0.797	0.701	0.923
	DAST-NDS	**0.857**	**0.818**	**0.900**
	DAST-D	0.857	0.818	0.900
LSI	*SAST-D*	0.739	0.658	0.842
	SAST-ConcD	**0.816**	**0.734**	**0.918**
	DAST-NDS	0.857	0.818	0.900
	DAST-D	0.857	0.818	0.900
KG	*SAST-D*	0.659	0.556	0.809
	SAST-ConcD	0.777	0.676	0.913
	DAST-NDS	0.727	0.667	0.800
	DAST-D	0.727	0.667	0.800

Table 3: Summary table of performance metrics (highlighted results show the best performing techniques for SAST and DAST).

4.2.1 Comparison of Semantic Similarity Techniques

Figure 1 and Figure 2 show the F-scores of different technique-corpus combinations over different similarity thresholds for SAST and DAST, respectively. We see that the F-scores increase with increasing similarity threshold, peaking at a threshold value ≥ 0.6 for DAST and at around 0.9 for SAST. Figure 3 in the appendix shows the performance metrics for clustering with knowledge graph-based semantic similarity. It is noteworthy that the F-scores for the knowledge graph-based clustering

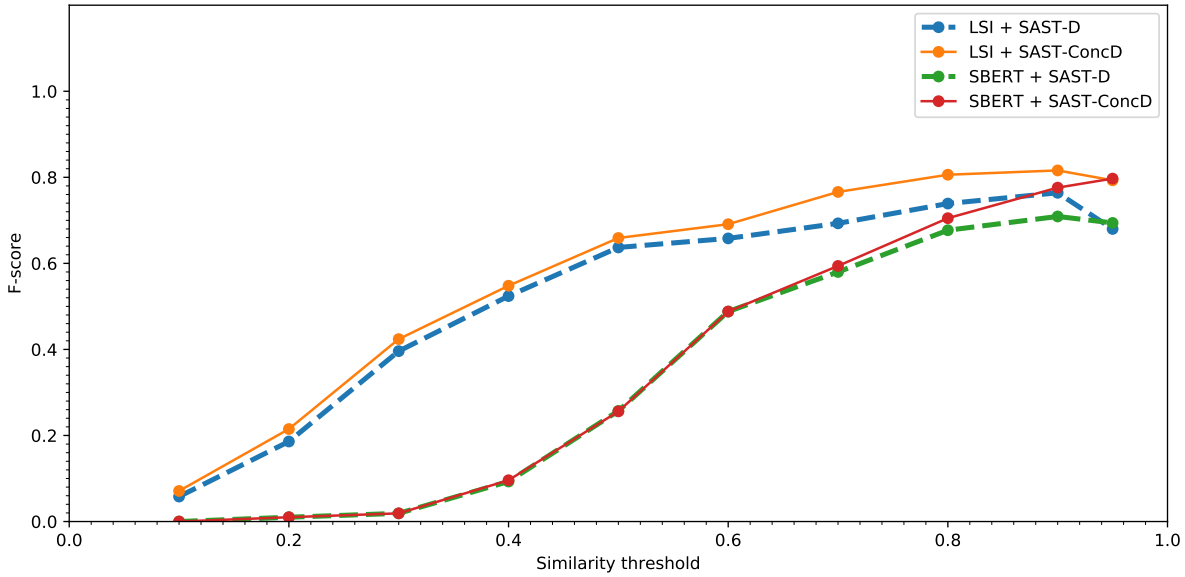

Figure 1: Semantic clustering results of SAST findings for different similarity thresholds.

are not only lower in comparison to *LSI* and *SBERT* but they also reach a plateau for threshold values higher than 0.2.

4.2.2 Qualitative Evaluation

For the qualitative evaluation, we showed incorrect predictions from the best results of semantic clustering of SAST and DAST findings to a security professional. The cluster results came from applying *LSI* to *SAST-ConcD* corpus for the SAST dataset and applying *SBERT* to *DAST-NDS* corpus for the DAST dataset. Using *SeFiLa*, the security professional inspected incorrect predictions and their associated ground-truth cluster. The security professional assigned possible reasons for poor duplicate identification by reading finding strings associated with incorrect predictions. These reasons are documented for 72 incorrect SAST predictions and 2 incorrect DAST predictions. The reasons and the number of times they were assigned to an incorrect prediction from either SAST or DAST clusters are listed in Table 4.

5 Discussion

From the quantitative evaluation, we see that SAST findings are best clustered by applying *LSI* to the *SAST-ConcD* corpus, which gives a F-score of 0.816. Although applying *SBERT* to the same corpora provides a similar F-score of 0.797 and matches a higher ratio of ground-truth clusters due to higher recall, *LSI* has a higher precision and less false positive predictions, which is a crucial requirement to the security professionals. Hence, applying *LSI* to *SAST-ConcD* corpus is our recommendation for identifying duplicate SAST findings.

When clustering DAST findings, we see that the highest F-score of 0.857 is achieved by applying *SBERT* and *LSI* to both *DAST-D* and *DAST-NDS* corpora. However, as illustrated in Figure 2, applying *SBERT* yields a high F-score for similarity threshold ≥ 0.6, whereas *LSI* yields a lower F-score. Since higher similarity thresholds are preferred in production scenarios to prevent false positive predictions, *SBERT* is preferred over *LSI*. For the corpus, *DAST-NDS* is preferred over *DAST-D* due to more textual content from three features, which leads to a better grasping of semantics and provides better distinction amongst false positives. We also see that for similarity threshold > 0.9, the F-score of *SBERT* with *DAST-NDS* slightly decreases. This is because of the strict distinction by semantic similarity algorithms, which also consider the semantics of a problem's solution when distinguishing between problems identified by different findings.

From the qualitative evaluation, we see that a significant challenge for SAST findings is the content of the finding description. Some tools provide a title instead of an actual description of the underlying problem. This leads to insufficient semantic content being derived from the finding corpus texts, thereby leading to poor duplicate identification. Another frequent reason for incorrect predictions in SAST are suboptimally constructed finding strings.

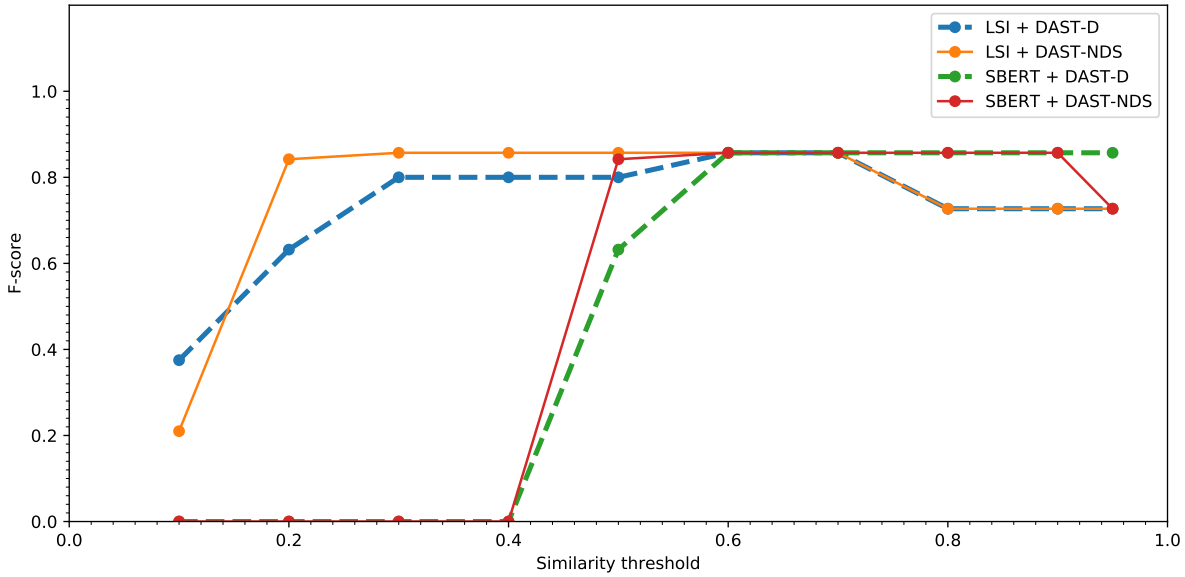

Figure 2: Semantic clustering results of DAST findings for different similarity thresholds.

This primarily arises when the information content of the original finding is low, which even is challenging for security professionals when determining duplicate findings just from reading the findings string. The third most highlighted challenge for SAST is the imbalance in the verbosity of description features of findings. The description either contains contextually rich problem descriptions or under-specified ones, leading to different extents of semantic content being captured and, thereby, incorrect comparisons being made. For DAST, we only have two incorrect predictions, which can be traced back to the challenge of being very application- and domain-specific.

To improve SAST findings clustering results, it is evident that the semantic content of finding strings representing a problem must be improved. This can be done by using multiple external sources, e.g., the National Vulnerability Database[4], the GitHub Advisory Database[5] for enrichment, or by scraping information from the multiple reference sources listed in a finding. The goal is that the textual data of each finding consists of multiple paragraphs and contains enough semantic content for the semantic similarity techniques to grasp as much contextual information as possible. Furthermore, the final corpus texts should contain the same verbosity level to avoid a bias related to the text length. Lastly, the same clustering approach can be studied using NLP models that are fine-tuned for security findings, accounting for the domain-specific vocabulary to improve the clustering results.

6 Limitations

While we present a variety of results regarding semantic clustering of security findings, our conclusions are limited in certain aspects. Firstly, all our findings result from scanning a single web application: JuiceShop. While it contains vulnerabilities encountered in real-world applications, it is restricted in its representation of a real scenario because JuiceShop is intended to comprise multiple vulnerabilities. Moreover, the subset of JuiceShop vulnerabilities that are clustered poorly might appear most often in reality, threatening the external validity of the results. Furthermore, our findings result from a finite number of modern security tools. While these tools are open-source and currently widely used, the scanning functionality of security testing tools is constantly evolving. Thereby, the scanning tools we use might change based on the needs of the domain. Lastly, our datasets were labeled by two security professionals and the results were evaluated by one security professional. While this is beneficial to prevent inconsistencies due to the subjective nature of the annotation tasks, the relevance of our results is highly dependent on the created ground-truth dataset. However, our chosen research design aims at making the results of our work as objective as possible. Researchers and

[4]https://nvd.nist.gov/
[5]https://github.com/advisories

Reason	Explanation for Incorrect Clustering	SAST	DAST
1	In the context of the product, this result can only be identified by somebody knowing the context of the application.	-	2
2	Different tools use a different phrasing to explain the same issue.	5	-
3	The tools sometimes provide no description of the finding. Hence, the features could only rely on the title.	39	-
4	Some tools provide more and some tools provide less text in their description, which reduces the impact of actual relevant features.	19	-
5	Additional review necessary due to an unknown reason for the decision.	5	-
6	The sub-optimally constructed feature string could be the reason for the incorrect clustering.	39	-
7	The tool describes the finding precisely according to the location of occurrence. Hence the finding text is over-specified.	3	-
8	Human annotation error and the suggested clustering by the algorithm is correct.	1	-
9	One tool addresses the issue of using an *eval* function, while the other one has the problem of user controlled values in it. However, it would not be considered as a major false positive.	3	-

Table 4: Overview of provided explanations from the qualitative evaluation.

practitioners can also use our developed annotation tool to reproduce our data collection or transfer our study insights to a setting of their own choice.

7 Conclusions and Future Work

In this work, we explored the applicability of semantic clustering of security findings through various similarity techniques. We tested three techniques from neural network-based, corpus-based, and knowledge-based methods on finding strings that describe security vulnerabilities identified by testing tools.

To this end, we created a ground-truth dataset of security findings clustered according to the expertise of security professionals. We compared this dataset to the results of semantic similarity techniques, indicating that SAST findings are best clustered by applying *LSI* to *SAST-ConcD* corpus, whereas DAST findings are best clustered by applying *SBERT* to *DAST-NDS* corpus. Conducting a qualitative evaluation with a security professional, we additionally pointed out the challenges encountered by semantic similarity techniques when applied to security findings and discussed possible solution strategies.

One potential future work would be the application of the chosen techniques to cluster security findings according to other testing strategies like solution-based clustering. This could grant deeper insights into the challenges of grouping security findings with NLP and provide access to new use cases. Furthermore, research on how plain neural networks perform when trained directly on semi-structured security findings appears to be promising given modern advancements in neural network architectures. Especially when compared to the NLP-based approach in this work, the properties of neural networks are worth exploring. Since neural networks automatically prioritize important features with layers like max-pooling, the manual effort undertaken to determine problem-describing features and clustering based on them could be alleviated. However, training a neural network requires significantly more data, so the construction of a much larger findings dataset would be necessary. Finally, an evaluation of the identified techniques in real-world DevOps scenarios could provide valuable insights into the practical usefulness of our approach in software development projects.

Acknowledgements

The authors want to thank the industry professionals for their exceptional effort in clustering the security findings and evaluating the shortcomings.

References

Karan Aggarwal, Finbarr Timbers, Tanner Rutgers, Abram Hindle, Eleni Stroulia, and Russell Greiner. 2017. Detecting duplicate bug reports with software engineering domain knowledge. *Journal of Software: Evolution and Process*, 29(3).

Dina Demner-Fushman and Jimmy Lin. 2006. Answer extraction, semantic clustering, and extractive summarization for clinical question answering. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 841–848, Sydney, Australia. Association for Computational Linguistics.

Lee R Dice. 1945. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302.

Hamzeh Eyal Salman, Mustafa Hammad, Abdelhak-Djamel Seriai, and Ahed Al-Sbou. 2018. Semantic clustering of functional requirements using agglomerative hierarchical clustering. *Information*, 9(9).

Mohammad Hossin and Md Nasir Sulaiman. 2015. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1.

Adrian Kuhn, Stéphane Ducasse, and Tudor Gîrba. 2007. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243. 12th Working Conference on Reverse Engineering.

Thomas K. Landauer and Susan T. Dumais. 1997. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211–240.

Olga Majewska, Diana McCarthy, Ivan Vulić, and Anna Korhonen. 2018. Acquiring verb classes through bottom-up semantic verb clustering. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

George A. Miller. 1995. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Phillip Schneider. 2020. App ecosystem out of balance: An empirical analysis of update interdependence between operating system and application software. Frankfurt University Library Johann C. Senckenberg.

Stacy Simpson. 2014. SAFECode whitepaper: Fundamental practices for secure software development 2nd edition. In *ISSE 2014 Securing Electronic Business Processes*, pages 1–32. Springer Fachmedien Wiesbaden.

Xi Tan, Xin Peng, Sen Pan, and Wenyun Zhao. 2011. Assessing software quality by program clustering and defect prediction. In *2011 18th Working Conference on Reverse Engineering*, pages 244–248.

A Supplementary Material

In this appendix, we provide additional material to the main article. Table 5 lists the security testing tools that were used to scan the web application JuiceShop and generate security findings. Figure 3 shows the performance metrics for clustering with knowledge graph-based semantic similarity.

Tool	Category	Analysis Type	Link
Anchore	SAST	Third-party vulnerabilities	anchore.com/opensource
Dependency Checker	SAST	Third-party vulnerabilities	owasp.org/dependency-check
Trivy	SAST	Third-party vulnerabilities	github.com/aquasecurity/trivy
GitLeaks	SAST	Hardcoded secrets	github.com/zricethezav/gitleaks
CodeQL	SAST	Coding flaws	codeql.github.com
Horusec	SAST	Coding flaws	horusec.io/site
Semgrep	SAST	Coding flaws	semgrep.dev
Arachni	DAST	Web app scan	github.com/Arachni/arachni
ZAP	DAST	Web app scan	www.zaproxy.org

Table 5: Overview of static (SAST) and dynamic (DAST) analysis security tools that were used to scan JuiceShop.

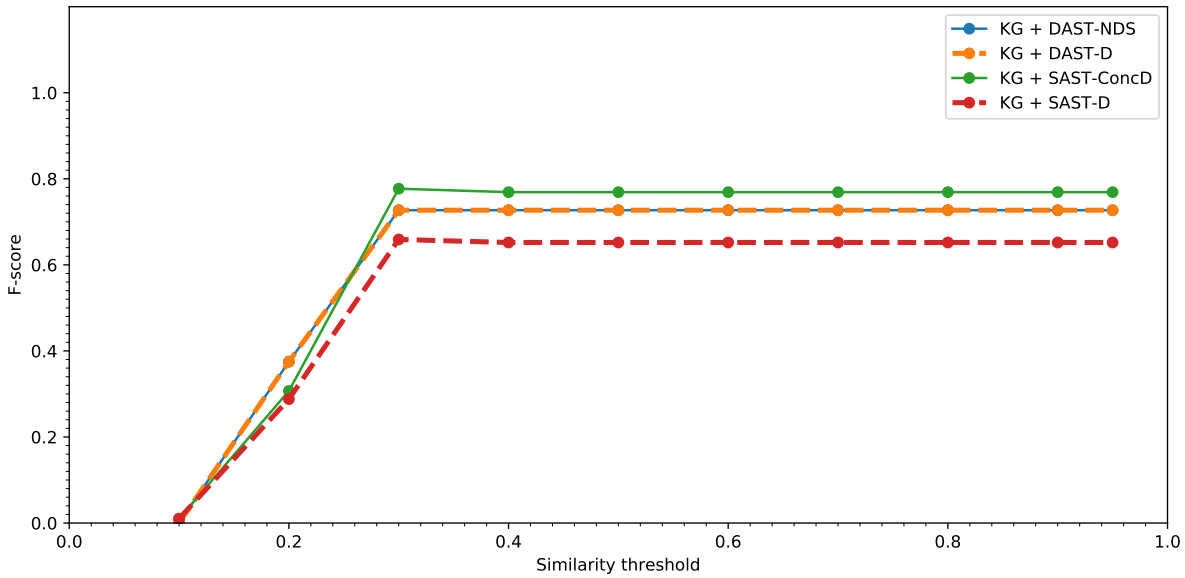

Figure 3: Semantic clustering results with knowledge graph-based similarity.