Does Faithful Data-Flow Access Reduce Lexical Reliance in Java Vulnerability Detection?

Urusha Shakhakarmi*, 장현지*, 김건민*, 김경백**

Abstract

Automated vulnerability detectors may achieve high accuracy while relying on superficial lexical cues rather than security-relevant program behavior. This study examines whether data-flow information reduces such reliance in java vulnerability detection. We compared three GraphCodeBert-based configuration, C1 uses lightweight dependency information with collapsed graph-guided attention, F1 uses the same dependency information with full graph-guided attention, and F2 uses a Tree-sitter based data flow graph with graph-guided attention. We mask security-related lexical cues while keeping the original structural information unchanges. F1 shows the lowest security-specific lexical effect (0.0058), compared with C1 (0.0182) and g2 (0.0167). The C1-F1 difference is statistically significant whereas C1 and F2 do not differ significantly. These results indicate that full graph-guided access can reduce lexical reliance in some configuration, but data-flow access alone does not consistently produce this effect.

Ⅰ. Introduction

Learning-based vulnerability detection increasingly incorporates program structure because vulnerabilities often depend on relationships among variables, statements, and API calls rather than on isolated source-code tokens. Earlier approaches mainly learned from lexical representations, while later methods incorporated control-flow,data-flow, and dependency information [1-2]. GraphCodeBERT integrates variable-level data-flow through graph-guided attention [3], and DeepDFA further demonstrates the data-flow information can improve vulnerability modeling [4]. These studies motivate the use of explicit structural information for vulnerability detection.

However, strong detection performance does not necessarily mean that a model bases its decisions on vulnerability-relevant behavior. Prior studies show that vulnerability detectors can exploit dataset-specific lexical artifacts, including variable names, function names, and other token patterns that correlate with labels [5-9]. This problem remains relevant even for structure-aware models. Providing a data-flow graph does not guarantee that the model reduces its reliance on such lexical cues.

Existing evaluations primarily measure predictive performance or compare models with and without structural information. These settings can show whether structural information affects predictive performance but they do not reveal whether access to structural information is associated with reduced
sensitivity to security-related lexical cues. In this study, we address this gap by keeping the original

Acknowledgement

* 전남대학교 일반대학원 인공지능융합학과 (대학원생, {urusha7403, gka1225, geonminkim}@jnu.ac.kr)
** 교신저자(corresponding author) : 전남대학교 일반대학원 인공지능융합학과 (교수, kyungbaekkim@jnu.ac.kr)

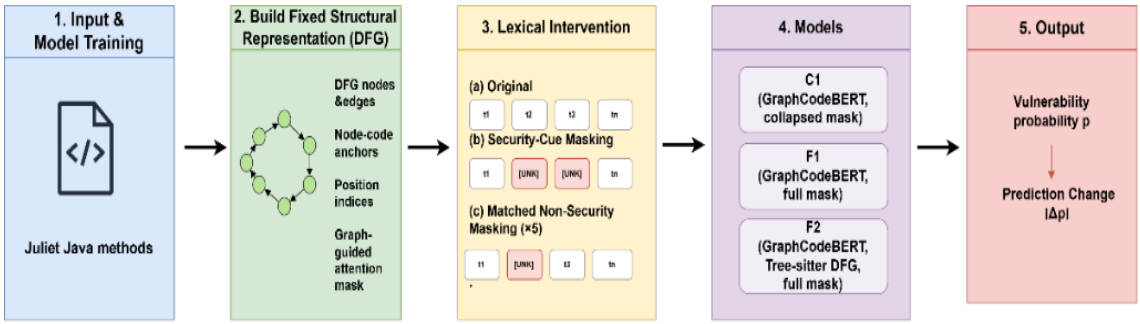

Fig 1 : Overall framework for evaluating the effect of data-flow access on security-related lexical reliance in Java vulnerability detection

data-flow representation fixed while selectively masking security-related lexical information. We compared C1, F1 and F2 to examine whether different graph-access and DFG-extraction stategies are associated with different levels of lexical reliance. This study provides a fixed-structure lexical intervention, compares three GraphCodeBERT-based structural configuration and uses matched non-security masking to isolate security-specific lexical sensitivity.

2. Methodology

2.1. Input and Model Preparation

We use Java methods from the Juliet benchmark [14], focusing on vulnerability categories associated with security-relevant data flow. Three GraphCodeBERT-based configurations are evaluated, C1, F1 and F2. Each model is trained independently on the original Juliet training set, and the resulting checkpoint is kept fixed during lexical analysis.

2.2. Fixed Structural Representation

For each test method, the original GraphCodeBERT feature representation such as code-token IDs, DFG nodes and edges, node-to-code line anchors, position indices, and graph guided attention mask are constructed and verified before lexical modification. Only selected code-token identities are modified. All DFG related structure (nodes, edges, anchors, attention mask) is reused unchanged across conditions ensuring that output differences arise from lexical changes rather than structure changes.

2.3. Lexical-Cue Intervention

Each eligible method is evaluated under three conditions. The unmodified original input, a security-cue masked condition in which security relevant identifiers are preserved during preprocessing are replaced with the model's placeholder token and a matched non-security masked in which non-security identifiers (excluding keywords, security-related identifiers and placeholder) are masked so that the number of removed subtokens exactly matches the security cue condition. This isolates the effect of removing security-related cues from general effect of removing lexical information. To reduce sampling variance, the matched control is repeated five times with independently sampled identifiers and averaged. Among 3,119 eligible methods 1,742 satisfied the intervention and were included in the final evaluation.

2.4. Lexical-Reliance Evaluation and Statistical Analysis

The original and masked inputs are evaluated using the same frozen checkpoints for each

model. We record the vulnerability probability and binary prediction for all conditions. Sensitivity is measured using absolute propbability change and prediction flip rate, while AUC summarize overall performance impact. Security-specific lexical reliance is defined as:

$$E(m,i) = D_{\text{sec}}(m,i) - D_{ctrl}(m,i)$$

where

$$D_{\text{sec}}(m,i) = \left| p_{\text{sec}}(m,i) - p_{\text{orig}}(m,i) \right|$$

and:

$$D_{ctrl}(m,i) = \frac{1}{K}\sum_{r=1}^{K} \left| p_{ctrl,r}(m,i) - p_{\text{orig}}(m,i) \right|, K = 5$$

A positive E indicates greater sensitivity to security-related cues than to an equivalent amount of non-security lexical removal. We estimate confidence intervals for mean E by resampling and assess significance using paired sign-flip permutation tests. For cross-model comparison, we evaluate matched pairs for C1-F1, C1-F2 and F1-F2 and apply Holm correction for multiple comparisons. C1-F1 is the primary comparison because the two methods use the same dependency extractor and differ only in graph-guided mask fidelity.

3. Result

Fig 2 shows the security-specific lexical effect E for C1, F1 and F2 while the original DFG information remains unchanged. F1 has the lowest effect (E=0.0058), compared with C1 (0.0182) and F2 (0.0167). This indicates lower lexical sensitivity for F1, while the similar values of C1 and F2 show that the reduction is not consistent across all DFG configurations.

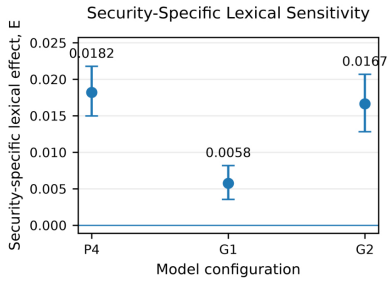

Fig 2: Security-Specific Lexical Effect Across Model Configuration

Table 1 summarizes performance and prediction changes under masking. All three models retain high AUC after security-cue masking, indicating only a small change in overall detection performance. However, security-cue masking produces larger probability shifts than matched non-security masking for every model. F1 shows the smallest security-related probability change (| Δp | sec=0.0078), compared with C1 (0.0195) and F2 (0.0202).

[Table 1]: Lexical-Intervention Performance and Probability-Shift Results

Model	Orig. Auc	Sec Auc	|Δp|sec	|Δp|ctrl	E [95% CI]
C1	0.981	0.981	0.0195	0.0013	0.0182 [0.0150, 0.0218]
F1	0.982	0.980	0.0078	0.0021	0.0058 [0.0035, 0.0082]
F2	0.981	0.977	0.0202	0.0036	0.0167 [0.0128, 0.0207]

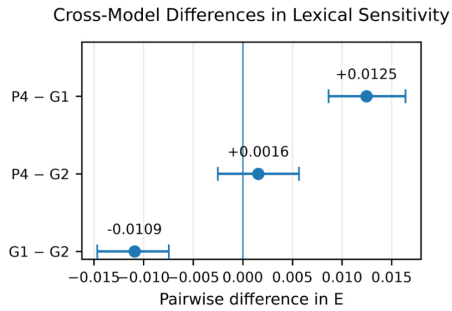

Fig 3 : Pairwise Differences in Security-Specific Lexical Effect

Fig 3 compares pairwise difference in E. The C1–F1 comparison shows a clear positive difference (ΔE = 0.0125) indicating that F1 is significantly less sensitive to security-related lexical cues than C1. The F1–F2 comparison (ΔE = 0.0109) also shows a clear difference while the C1–F2 difference (ΔE = 0.0016) is small with a confidence interval spanning zero. Overall, lower lexical reliance depends on model configuration and is not guaranteed by DFG alone.

4. Discussion and Conclusion

The result shows that the lexical reliance varies across the three structural configuration, even though the original DFG remains unchanged. F1 shows significantly lower security-specific lexical sensitivity than C1, indicating that full graph-guided access reduces lexical dependence when the same lightweight dependency extractor is used. F2, however, does not show this reduction and remains similar to C1. These findings suggest that data-flow access alone does not consistently reduce lexical reliance. Instead, the effect may depend on both graph access and the underlying dependency representation. This study is limited to eligible Juliet methods and three GraphCodeBERT-based configuration. In future work, we plan to investigate whether these findings generalize across a broader range of Java datasets and models.

Reference

[1] Zhou, Y., Liu, S., Siow, J., Du, X. and Liu, Y., 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. Advances in neural information processing systems, 32.

[2] Li, Y., Wang, S. and Nguyen, T.N., 2021, August. Vulnerability detection with fine-grained interpretations. In Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering (pp. 292–303).

[3] Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S. and Tufano, M., 2020. Graphcodebert: Pre-training code representations with data flow. arXiv preprint arXiv:2009.08366.

[4] Steenhoek, B., Gao, H. and Le, W., 2024, February. Dataflow analysis-inspired deep learning for efficient vulnerability detection. In Proceedings of the 46th ieee/acm international conference on software engineering (pp. 1–13).

[5] Chakraborty, S., Krishna, R., Ding, Y. and Ray, B., 2021. Deep learning based vulnerability detection: Are we there yet?. IEEE Transactions on Software Engineering, 48(9), pp.3280–3296.

[6] Croft, R., Babar, M.A. and Kholoosi, M.M., 2023, May. Data quality for software vulnerability datasets. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (pp. 121–133). IEEE.

[7] Ding, Y., Fu, Y., Ibrahim, O., Sitawarin, C., Chen, X., Alomair, B., Wagner, D., Ray, B. and Chen, Y., 2024. Vulnerability detection with code language models: How far are we?. arXiv preprint arXiv:2403.18624.

[8] Yefet, N., Alon, U. and Yahav, E., 2020. Adversarial examples for models of code. Proceedings of the ACM on Programming Languages, 4(OOPSLA), pp.1–30.

[9] Rahman, M.M., Ceka, I., Mao, C., Chakraborty, S., Ray, B. and Le, W., 2024, April. Towards causal deep learning for vulnerability detection. In Proceedings of the IEEE/ACM 46th international conference on software engineering (pp. 1–11).

[10] Cao, S., Sun, X., Wu, X., Lo, D., Bo, L., Li, B. and Liu, W., 2024, April. Coca: Improving and explaining graph neural network-based vulnerability detection systems. In Proceedings of the IEEE/ACM 46th international conference on software engineering (pp. 1–13).