Detecting Vulnerabilities from Issue Reports for Internet-of-Things

1st Sogol Masoumzadeh
Electrical and Computer Engineering
McGill University
Montreal, Canada
sogol.masoumzadeh@mail.mcgill.ca

***Abstract*—Timely identification of issue reports reflecting software vulnerabilities is crucial, particularly for Internet-of-Things (IoT) where analysis is slower than non-IoT systems. While Machine Learning (ML) and Large Language Models (LLMs) detect vulnerability-indicating issues in non-IoT systems, their IoT use remains unexplored. We are the first to tackle this problem by proposing two approaches: (1) combining ML and LLMs with Natural Language Processing (NLP) techniques to detect vulnerability-indicating issues of 21 Eclipse IoT projects and (2) fine-tuning a pre-trained BERT Masked Language Model (MLM) on 11,000 GitHub issues for classifying vulnerability-indicating issues. Our best performance belongs to a Support Vector Machine (SVM) trained on BERT NLP features, achieving an Area Under the receiver operator characteristic Curve (AUC) of 0.65. The fine-tuned BERT achieves 0.26 accuracy, emphasizing the importance of exposing all data during training. Our contributions set the stage for accurately detecting IoT vulnerabilities from issue reports, similar to non-IoT systems.**

***Index Terms*—Vulnerabilities, Issue Trackers, IoT, Machine Learning, Fine-tuning.**

I. Introduction

Motivation. Submissions to software issue trackers describe bugs, desirable features, and enhancement requests that software teams need to prioritize [1]. Popular projects attract a large volume of issue reports, making it a challenging task for developers to prioritize issues based on their risk level. Consequently, severe issues such as software vulnerabilities may not be prioritized, endangering dependent products [2]. This problem is exacerbated in *Internet-of-Things (IoT)*. IoT systems are heterogeneous ecosystems that are composed of loosely interconnected hardware, middleware, and software nodes [3]. Compared to other software, code bases in IoT systems are more prone to cross-cutting defects [4]. In turn, issues reflecting vulnerabilities are vague and verbose, prolonging their reasoning even more [5]. Hence, automated identification of such issues that report on vulnerabilities in IoT systems (i.e., *vulnerability-indicating issues*) is of great value.

Problem. Conventional *Machine Learning (ML)* models [6] and autoregressive *Large Language Models (LLMs)* [7] are explored for automatically classifying vulnerability-indicating issues in web browsers, cloud, and databases [8]–[12]. Meanwhile, the application of these models for detecting IoT vulnerability-indicating issues is yet to be investigated.

Background and related work. Peters et al. [9] filtered and ranked $45,940$ issues of Chromium, Wicket, Ambari, Camel, and Derby to remove non-security issues with security-related tags for enhancing classification performances of text-based models. Das et al. [11] used ML models and probabilistic *Natural Language Processing (NLP)* techniques to classify issues from [9] as security or non-security related.

II. Approach and Uniqueness

We are the **first, initiating the detection of vulnerabilities in IoT systems from their issue reports**. We propose an empirical study, illustrated in Figure 1, in which we evaluate the performance of ML models that leverage NLP techniques (16 combinations), and OpenAI's *gpt-4o-1106-preview (GPT-4o)* [13] to detect vulnerability implications in a corpus of issues from *Eclipse IoT* open-source projects. Additionally, we fine-tune a pre-trained neural *Masked Language Model (MLM)* on an extensive corpus of unlabeled GitHub issues [14], on the downstream task of detecting vulnerability-indicating issues.

A. The empirical study

Subjects. From Eclipse IoT projects, we consider active projects with executable CI/CD pipeline and integrated static or dynamic analyzers. 21 projects satisfy our inclusion criteria.

Corpus. We mine GitHub resolved issues of the 21 Eclipse IoT projects (i.e., $9,564$ reports). From these issues, we randomly sample 370, providing 95% Confidence Level (CL) and ±5% Confidence Interval (CI). We then expand the corpus by applying stratified sampling, adding 50 new issues each round, until the classification performance saturates (Figure 2) at 820 issues, with 63 as vulnerability-indicating.

Labeling. We label the issues as either vulnerability-indicating or non-vulnerability-indicating based on whether they hint at an exploitable vulnerability in an IoT context. The labeling is carried out by three inspectors until consensus.

Pre-processing. We pre-process the corpus to remove special characters, English stop words, and memory traces, replace execution logs with semantically equivalent natural text, and lemmatize inflicted terms.

Training. We evaluate the performance of *Gaussian Naïve Bayes (GNB)* [15], *Support Vector Machine (SVM)* [16], *Random Forest (RF)* [17], and *Logistic Regression (LR)* [18] in classifying vulnerability-indicating issues through *10-fold cross-validation (10-CV)*. We follow the *grid search algorithm* to optimize the classifier hyper-parameters. We vectorize our

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00394

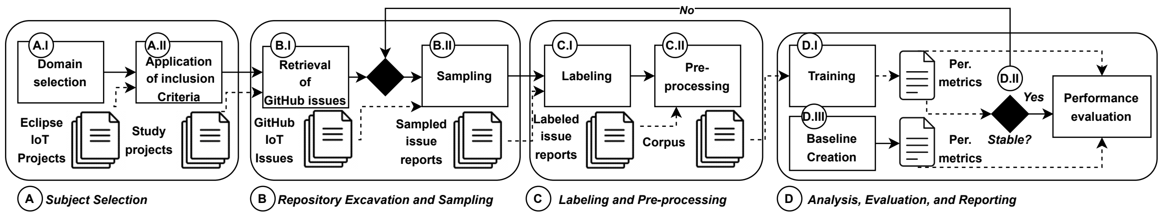

Fig. 1: The overview of the study

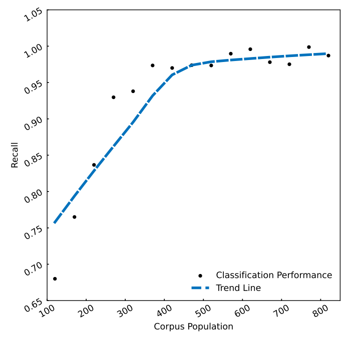

Fig. 2: The saturation of the classification performance

corpus using *BoW* [19], *BERT* [20], *GloVe* [21], and *W2V* [22] NLP features. To balance out the quantities of data classes, we re-sample the corpus by increasing the population of vulnerability-indicating issues. Additionally, after carefully handcrafting the prompts, we instruct GPT-4o to classify vulnerability-indicating issues. To account for the LLM's response stochasticity, we set the *Seed* decoding parameter as a constant value. Additionally, we measure the model's performance using *pass@3* which is the upmost correct responses given three generation attempts per issue [23].

Evaluation. We evaluate the performance of our models against a *random guesser* with the *Area Under the receiver operator characteristic Curve (AUC)* of 0.50.

B. The fine-tuning experiment

Corpus. We mine GitHub resolved issues if they are not a pull request, if both title and submission description are non-empty, and if their submission date is between 2022-01-01 and 2024-03-01. From issues satisfying these criteria, those tagged as "security" are included as vulnerability-indicating ($6,696$ issues). Issues tagged as "bug" but not as "security" are included as non-vulnerability-indicating ($528,494$ issues). From these issues, we randomly sample $11,000$ with stratification (95% CL and $\pm 5\%$ CI) as our corpus.

The Surrogates. For each issue label, we retrieve a list of semantically similar, substitution keywords which we call the *Surrogates*. To do so, (1), we pre-process the corpus by performing the same steps discussed in Section A. Afterwards (2), for each label, we run the *Rapid Automatic Keyword Extraction (RAKE)* [24] algorithm to determine and rank keywords by calculating frequencies of words in each category and analyzing their co-occurrences with the rest of the context. Then (3), for each label, we manually evaluate the top-100 keywords to remove those irrelevant to security-related topics. Finally (4), we discard keywords that appear for both labels. We set the Surrogates as the top-10 final refined keywords for each label. The Surrogates for vulnerability-indicating and non-vulnerability-indicating labels are: [*CVE*, *GitHub*, *version*, *use*, *vulnerability*, *issue*, *security*, *severity*, *NVD*, and *check*] and [*data*, *file*, *foundry*, *type*, *filter*, *fail*, *error*, *debug*, *default*, and *warn*], respectively.

MLM training. We fine-tune a BERT MLM, hiding occurrences of the Surrogates in the corpus, using the [MASK] token. 10-CV, with training epochs=2 and batch size=5, is executed on a NVIDIA T4 hardware accelerator.

TABLE I: The empirical study results (AUC values)

	BoW	BERT	GloVE	W2V
GNB	0.53	0.62	0.54	0.55
SVM	0.55	**0.65**	0.61	0.58
RF	0.45	0.63	0.58	0.60
LR	0.58	0.59	0.45	0.44

III. Results and Contributions

Performances of ML models in classifying vulnerability-indicating issues are demonstrated in Table I. For the same experiment, GPT-4o achieves an AUC of 0.60. Our most accurate classifier is the SVM trained using BERT NLP features and achieves an AUC of 0.65. On the other hand, the mean classification accuracy of the fine-tuned BERT MLM is only 0.26 across ten training folds. This considerable difference between the performances of the classification settings highlights the importance of exposing all the context to the classifiers while training. Although the corpus of the fine-tuning experiment is much more extensive compared to the corpus of the empirical study, masking the Surrogates prevents the model to learn the entire data: (1) many issues do not contain any of the Surrogates and hence are not exposed to BERT MLM and (2) the model is not trained on the [CLS] token to predict the label while seeing the whole issue description.

Future work. We plan to add an unsupervised training layer that exposes BERT MLM to all issues, allowing it to iteratively refine its prediction generalizability. We also aim to extend the experiments with additional baselines and larger datasets.

References

[1] I. Mistrík, J. Grundy, A. van der Hoek, and J. Whitehead, "Collaborative Software Engineering: Challenges and Prospects," in *Collaborative Software Engineering*. Springer, 2010, pp. 389–403.

[2] Z. Durumeric *et al.*, "The matter of heartbleed," in *The 2014 Internet Measurement Conference*. ACM, 2014, pp. 475–488.

[3] S. Li *et al.*, "The internet of things: A survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[4] N. Khezemi, S. Ejaza, N. Moha, and Y.-G. Guéhéneuc, "Comparison of code quality and best practices in iot and non-iot software," *arXiv preprint arXiv:2408.02614*, 2024.

[5] VDC Research Group and Memfault, "The state of iot software development: A benchmark for embedded teams and leaders," https://go.memfault.com/hubfs/The%20State%20of%20IoT%20-%20VDC%20Memfault%20Whitepaper%202024.pdf, VDC Research Group, Global Survey of 783 IoT/embedded developers, Technical Report April 2024, Apr. 2024, 50.3% take over a week to find defects; 19.6% take several months; 40%+ take over a week to fix once found.

[6] R. Kallis, A. Di Sorbo, G. Canfora, and S. Panichella, "Ticket tagger: Machine learning driven issue classification," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 406–409.

[7] B. Min, H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth, "Recent advances in natural language processing via large pre-trained language models: A survey," *ACM Comput. Surv.*, vol. 56, no. 2, 2023.

[8] D. Behl, S. Handa, and A. Arora, "A bug mining tool to identify and analyze security bugs using naive bayes and tf-idf," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. IEEE, 2014, pp. 294–299.

[9] F. Peters, T. T. Tun, Y. Yu, and B. Nuseibeh, "Text filtering and ranking for security bug report prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 6, pp. 615–631, 2017.

[10] D. Zou, Z. Deng, Z. Li, and H. Jin, "Automatically identifying security bug reports via multitype features analysis," in *Information Security and Privacy: 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings 23*. Springer, 2018, pp. 619–633.

[11] D. C. Das and M. R. Rahman, "Security and performance bug reports identification with class-imbalance sampling and feature selection," in *2018 Joint 7th International Conference on Informatics, Electronics & Vision (ICIEV) and 2018 2nd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE, 2018, pp. 316–321.

[12] S. Mostafa, B. Findley, N. Meng, and X. Wang, "Sais: Self-adaptive identification of security bug reports," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 1779–1792, 2019.

[13] O. Team, "Chatgpt: Optimizing language models for dialogue," 2022.

[14] Y. Meng, Y. Zhang, J. Huang, C. Xiong, H. Ji, C. Zhang, and J. Han, "Text classification using label names only: A language model self-training approach," *arXiv preprint arXiv:2010.07245*, 2020.

[15] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *The 2001 Workshop on Empirical Methods in Artificial Intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[16] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data," *Bioinformatics*, vol. 16, no. 10, pp. 906–914, 2000.

[17] G. Biau and E. Scornet, "A random forest guided tour," *Test*, vol. 25, no. 2, pp. 197–227, 2016.

[18] R. E. Wright, "Logistic regression." *Reading and Understanding Multivariate Statistics*, pp. 217–244, 1995.

[19] W. A. Qader, M. M. Ameen, and B. I. Ahmed, "An overview of bag of words;importance, implementation, applications, and challenges," in *The 2019 International Engineering Conference*, 2019, pp. 200–204.

[20] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *The 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[21] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *The 2014 Conference on Empirical Methods in Natural Language Processing*. ACL, 2014, pp. 1532–1543.

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *The 1st International Conference on Learning Representations*, 2013.

[23] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[24] S. Rose, D. Engel, N. Cramer, and W. Cowley, "Automatic keyword extraction from individual documents," *Text mining: applications and theory*, pp. 1–20, 2010.