Vulnerable Line Detector
DD2421 Machine Learning

Sofia Bobadilla
KTH Royal Institute of Technology
School of Electrical Engineering and Computer Science, Theoretical Computer Science

March 7, 2025

1 Overview

This report presents the final project for the graduate course on machine learning DD24421. Per instructions it is related to the student's PhD topic: smart contract security.

In that sense, this project dives into vulnerable code detection for smart contracts. The problem of vulnerability detection for smart contracts has been vastly covered by previous work [1, 2] and therefore this project will aim to navigate the impact of training labels on a machine learning-based detector for vulnerable code.

Recent work on the use of machine learning for vulnerability detection, proposes that mislabeled datasets on training affects tremendously a model's performance when tested on a realistic dataset [3]. IN the context of smart contract, a new treand has arrived on vulnerability detection for smart contracts motivating that a true positive vulnerability should have a corresponding exploit [4, 5].

This project will test that idea by training two models in different versions of a vulnerable smart contract dataset. The first Model, later called Mix-Model(MM), will train on lines of code labeled as vulnerable by the original dataset information. The second model, later called Pure-Model(PM), will train on the lines of code from the same dataset, yet the labels will be modified by the exploitability of the corresponding contracts.

The dataset selected for the first model is the well-known smartbugs-curated dataset [6], and the dataset for the training of the second model will be a curated version of it from the exploitability information, of smartbugs-curated, collected by sb-heists [5].

The first model (MM) is trained on lines of code labeled as vulnerable by smartbugs-curated, and non-vulnerable lines randomly selected from the solidity files in the dataset. The second model (PM) is trained on vulnerable lines of code that belong to a contract that is confirmed to be exploitable by sb-heists, and as non-vulnerable lines considers the lines of code that do not have an associated exploit from sb-heists.

The evaluation will consider each model's performance on each dataset. Particularly interesting will be the recall from MM in PM's testing dataset.

2 Context

Recent surveys on the effectiveness of vulnerability detection for smart contracts show poor performance across a wide selection of tools [7, 1, 8]. Particularly, there has been a new perspective on the problem by connecting vulnerabilities and exploits [4, 5].

From the security perspective of smart contracts, a True Positive vulnerability has an exploit that will lead to a financial loss [5]. This financial loss can come in the form of leak Ether, due to oracle manipulation, denial of service, or others. Still, this connection between vulnerable and exploitable has never been considered before as a way to improve the performance of machine learning-based detectors.

Recent work proposes a new way to validate smart contract automated fixes by validating its ability to mitigate and exploit [5]. The authors use the well-known dataset smartbugs curated and manually crafted exploits for each smart contract on the dataset. Their results show that a significant portion of the dataset was indeed non-exploitable.

The work of Chakraborty, et al. [3] delves into the performance of deep learning-based vulnerability detection on realistic datasets. Definning realistic datasets as datasets with more accurate labelling.

This project aims to study the impact of such a curation process on the training dataset on machine learning-based approaches for vulnerability detection.

3 Methodology

3.1 Dataset

The project relies on two previous works on smart contract vulnerability for labeling. The contracts and labels for the first model are obtained from the dataset smartbugs-curated [1]. The curation process will used the exploitability information from sb-heists [2]

3.1.1 smartbugs-curated

The dataset comprises 143 vulnerable smart contracts, each annotated with one of ten categories derived from the DASP taxonomy: reentrancy;access control;arithmetic;unchecked low-level calls; denial of services;bad randomness;front running;time manipulation;short addresses;miscellaneous.

3.1.2 sb-heists

sb-heists is a framework for evaluating smart contract security patches against real-world exploits including a reproducible dataset of smart contract exploits derived from the smartbugs-curated dataset. This project provides the list of contracts that have a corresponding exploit[3], 91 in total. It is worth nothing that the evaluation of the contracts and the exploits correspond to a manual effort of the authors.

3.2 Data Encoding

To encode the data points from both datasets, I considered the following criteria:

- **Code Encoding**

[1] `https://github.com/smartbugs/smartbugs-curated/`
[2] `https://github.com/ASSERT-KTH/sb-heists`
[3] https://github.com/ASSERT-KTH/sb-heists/blob/main/not-exploitable.md

1 `code_enc`: Encodes each unique Solidity code snippet using `LabelEncoder`, assigning a unique integer to each distinct snippet.

- **Token Count**

2 `tokens`: Tokenizes Solidity code by removing comments and splitting on non-alphanumeric characters. The total count of tokens is stored as a numerical feature.

- **Presence of External Calls**

3 `has_external_call`: Checks if the Solidity code contains low-level function calls like `call`, `delegatecall`, `staticcall`, `send`, or `transfer`. Stores this as a binary feature (1 if present, 0 otherwise).

- **Presence of Require or Assert**

4 `has_require_assert`: Checks if the Solidity code contains the `require` or `assert` statements. Stores this as a binary feature (1 if present, 0 otherwise).

- **Pragma Version Encoding**

5 `pragma_major`: Extracts and encodes the major version from the Solidity `pragma` statement as an integer.

6 `pragma_minor`: Extracts and encodes the minor version from the Solidity `pragma` statement as an integer.

7 `pragma_patch`: Extracts and encodes the patch version from the Solidity `pragma` statement as an integer.

- **Line Number Normalization**

8 `line_number`: Normalizes the line number using `MinMaxScaler` to scale values between 0 and 1.

- **Label Encoding**

9 `label`: Converts the vulnerability label into an integer (0 or 1).

3.3 Classifier

The classifier used in this code is a Random Forest model, which is an ensemble learning method that builds multiple decision trees and aggregates their outputs to improve performance and reduce overfitting. Here's a breakdown of its configuration and training process:

3.3.1 Classifier Description

For both experiments, the classifier considers the following configuration.

- Algorithm: Random Forest (ensemble of decision trees)
- Hyperparameters:
  1. max_depth=10: Limits the depth of each tree to prevent overfitting.
  2. min_samples_leaf=1: A leaf node must have at least one sample.
  3. min_samples_split=10: A node must have at least 10 samples to be split further.
  4. n_estimators=300: Uses 300 trees in the forest for better stability.
  5. random_state=42: Ensures reproducibility.

3.3.2 Labeling

Both classifier will aim at detecting vulnerable line of code, independent of the type. This will be presented with a `1` if the line of code is vulnerable and `0` if not.

3.4 Mix Model (MM)

MM will train using the labels from the smartbugs-curated dataset.

- Dataset: smartbugs-curated
- Labels: smartbugs-curated
- Train data: 355 samples
  - vulnerable: 178
  - non-vulnerable: 177
- Test data: 89 samples
  - vulnerable: 44
  - non-vulnerable: 45

3.5 Pure Model (PM)

PM will train on lines of code that are label as vulnerable if they are reported as vulnerable in the smartbugs-curated dataset and the contract has been listed as exploitable by sb-heists.

- Dataset: smartbugs-curated
- Labels: smartbugs-curated x sb-heists (exploits)
- Train data: 177 samples
  - vulnerable: 106
  - non-vulnerable: 71
- Test data: 45 samples
  - vulnerable: 27
  - non-vulnerable: 18

3.6 Research Question

The research question for this project, motivated by [3], is:

to what extent the mislabeled vulnerabilities in the training data of a model for vulnerability detection can affect its performance on realistic datasets?

4 Results

The results consist of the metrics of Accuracy, Precision and Recall of both models, MM and PM, on both testing dataset. The results are shown in Fig 4

The MM model achieves an accuracy of 0.887, a precision of 0.84, and a recall of 0.954 on the SmartBugs-Curated dataset. On the SB-Heists X SmartBugs-Curated dataset, its accuracy decreases to 0.42, with a precision of 0.518 and a recall of 0.518.

The PM model attains an accuracy of 0.426, a precision of 0.44, and a recall of 0.659 on the SmartBugs-Curated dataset. On the SB-Heists X SmartBugs-Curated dataset, its accuracy increases to 0.844, with a precision of 0.833 and a recall of 0.925.

Dataset	MM			PM		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
smartbugs-curated	0.887	0.84	0.954	0.426	0.44	0.659
sb-heists X smartbugs-curated	0.42	0.518	**0.518**	0.844	0.833	**0.925**

Table 1: Performance metrics for MM and PM models

5 Discussion

The results provide insight into how different labeling strategies affect the performance of machine learning models for smart contract vulnerability detection. By comparing the MM and PM models, trained on different label definitions, we can observe the extent to which training on a dataset with purely syntactic vulnerability annotations (MM) generalizes to a dataset where vulnerabilities are tied to real-world exploits (PM).

The evaluation particularly highlights differences in recall, which is crucial in security applications where missing a true vulnerability can have severe consequences.

6 Resources

Please refer to the following links to find the resources used in this assignment:

1. repository
2. MM's dataset
3. PM's dataset
4. data processing
5. classifier and experiments

References

[1] Monika di Angelo et al. "Evolution of automated weakness detection in Ethereum bytecode: a comprehensive study". In: *Empirical Software Engineering* 29.2 (2024), p. 41. ISSN: 1573-7616. DOI: 10.1007/s10664-023-10414-8. URL: https://doi.org/10.1007/s10664-023-10414-8.

[2] Erfan Andesta, Fathiyeh Faghih, and Mahdi Fooladgar. "Testing Smart Contracts Gets Smarter". In: *2020 10th International Conference on Computer and Knowledge Engineering (ICCKE)*. 2020, pp. 405–412. DOI: 10.1109/ICCKE50421.2020.9303670.

[3] Partha Chakraborty et al. "Revisiting the Performance of Deep Learning-Based Vulnerability Detection on Realistic Datasets". In: *IEEE Transactions on Software Engineering* 50.8 (2024), pp. 2163–2177. DOI: 10.1109/TSE.2024.3423712.

[4] Tianyuan Hu et al. "Why Smart Contracts Reported as Vulnerable Were Not Exploited?" In: *IEEE Transactions on Dependable and Secure Computing* (2024), pp. 1–18. DOI: 10.1109/TDSC.2024.3520554.

[5] Sofia Bobadilla, Monica Jin, and Martin Monperrus. *Do Automated Fixes Truly Mitigate Smart Contract Exploits?* 2025. arXiv: 2501.04600 [cs.SE]. URL: https://arxiv.org/abs/2501.04600.

[6] Monika di Angelo et al. "SmartBugs 2.0: An Execution Framework for Weakness Detection in Ethereum Smart Contracts". In: *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)*. to appear. 2023.

[7] Thomas Durieux et al. "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE '20. Seoul, South Korea: Association for Computing Machinery, 2020, pp. 530–541. ISBN: 9781450371216. DOI: 10.1145/3377811.3380364. URL: https://doi.org/10.1145/3377811.3380364.

[8] C. Sendner et al. "Large-Scale Study of Vulnerability Scanners for Ethereum Smart Contracts". In: *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2024, pp. 220–220. DOI: 10.1109/SP54263.2024.00182. URL: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00182.