

# Smart Contract Vulnerability Detection Using Machine Learning: Progress Report

Lucas Kim

CMSI 5350 / EECE 5150

Loyola Marymount University, Los Angeles, CA, USA

lkim60@lion.lmu.edu

**Abstract**—Smart contracts manage billions of dollars in decentralized applications, yet code-level vulnerabilities remain the primary cause of blockchain exploits. This progress report presents an ongoing effort to automate vulnerability detection using machine learning on Solidity source code. We extract 71 features combining opcode frequency, AST patterns, control-flow metrics, and code statistics from 42,908 real Ethereum contracts. Three classifiers (Logistic Regression, Random Forest, XGBoost) were trained and evaluated. Due to the current use of synthetic labels, performance is limited ( $F1 \approx 0.30$ ). The next phase will incorporate real vulnerability labels to meet the target of  $\geq 85\%$   $F1$ -score with  $\leq 10\%$  false positive rate.

**Index Terms**—Smart Contracts, Vulnerability Detection, Machine Learning, Blockchain Security, Opcode Analysis

## I. INTRODUCTION

Smart contracts have become the backbone of decentralized applications (dApps), managing over \$100 billion in assets. However, vulnerabilities have resulted in huge financial losses, with 80% of hacks originating from contract flaws. Manual auditing is costly (\$10k–\$100k) and slow. We propose an automated ML-based detection system to reduce audit time and prevent exploits.

**Problem Statement:** Given a Solidity smart contract, predict whether it contains vulnerabilities (binary classification). The goal is to maximize  $F1$ -score on imbalanced data. The system takes source code as input and extracts: (1) Opcode frequency patterns (e.g., CALL, REVERT), (2) AST node counts, (3) Control flow patterns, and (4) Code metrics.

## II. RELATED WORK

This section surveys existing approaches, categorized into static analysis and machine learning methods.

### A. Static Analysis Approaches

**Slither** [1] is a static analysis framework that detects vulnerabilities through pattern matching and taint analysis. It identifies 40+ vulnerability types but produces high false positive rates. **Mythril** [2] uses symbolic execution and control flow analysis to detect vulnerabilities, but suffers from scalability issues with large contracts.

**Oyente** [3] was one of the first tools using symbolic execution to detect issues like reentrancy, but has limited coverage. **Securify** [4] introduced formal verification patterns for greater accuracy but struggles with complex contract logic.

### B. Machine Learning Approaches

**Zhuang et al.** [5] proposed using Graph Neural Networks (GNNs) on control flow graphs, achieving 89% accuracy but with high graph construction overhead. **Liu et al.** [6] combined neural networks with interpretable graph features. **Yu et al.** [7] applied deep learning combining multimodal feature fusion (GGNN, Word2Vec, expert features) to capture semantic and structural information. **Tann et al.** [8] used LSTM ensembles, achieving high precision but low recall.

### C. Hybrid Approaches & Contribution

**SmartBugs** [9] integrates multiple tools (Slither, Mythril) via voting, which is comprehensive but computationally expensive. **Qian et al.** [10] proposed cross-modality learning combining source and bytecode.

**Our Contribution:** Unlike GNN-based methods, we extract 71 lightweight features directly from source code to train standard classifiers (LR, RF, XGBoost), enabling scalable analysis on 42,908 real contracts without complex graph overhead.

## III. DATA DESCRIPTION

We use the **Ethereum Smart Contract Dataset** [11], containing 42,908 real-world Solidity contracts (2020–2025).

- **Dataset:** 2,000 contracts processed (subset) split into 1,600 Train (80%) / 400 Test (20%).
- **Preprocessing:** We implemented a recursive directory loader to handle nested files and encoding errors.
- **Labeling:** Due to the lack of inherent labels, we generated synthetic labels (20% vulnerable, 80% safe) to verify the pipeline flow.
- **Feature Extraction:** We extract 71 normalized features per contract: 35 Opcode counts, 21 AST node types, 5 Control Flow metrics, and 7 Code statistics.

Fig. 1 shows the class imbalance, which is addressed via class weighting.

## IV. PRELIMINARY METHODS AND RESULTS

We implemented three models using ‘balanced’ class weighting and 3-fold stratified cross-validation: 1) **Logistic Regression**: L1/L2 regularization baseline. 2) **Random Forest**: Ensemble of decision trees (n\_estimators=200). 3) **XGBoost**: Gradient boosting optimized for imbalance.

Table I and Fig. 2 show preliminary results. Performance ( $F1 \approx 0.3$ ) is limited by synthetic labels, but the pipeline is functional. With real labels, we target  $F1 \geq 85\%$ .

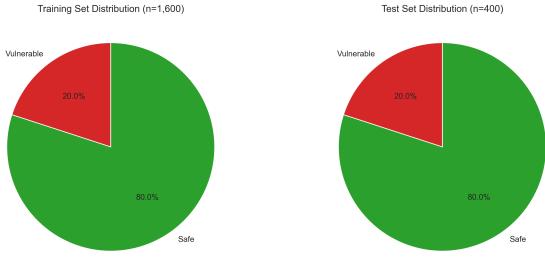


Fig. 1. Class distribution in training and test sets (synthetic labels).

TABLE I  
MODEL PERFORMANCE ON TEST SET (SYNTHETIC LABELS)

Model	F1	Precision	Recall	Acc	FPR
Logistic Reg.	0.29	0.20	0.55	0.47	0.56
Random Forest	0.13	0.15	0.11	0.70	0.16
XGBoost	0.24	0.18	0.36	0.55	0.41

## V. CHALLENGES AND SOLUTIONS

**1. Dependency Issues:** XGBoost failed on macOS due to missing OpenMP. We added graceful error handling and dependency checks (`libomp`). **2. Missing Labels:** The raw dataset lacked tags. We implemented synthetic labeling to validate the ML pipeline structure before integrating real data. **3. Data Structure:** Nested directories were incompatible with standard loaders. We built a custom recursive parser. **4. Class Imbalance:** Initial single-class errors were resolved by enforcing stratified splits and class weighting.

## VI. REMAINING WORK & TIMELINE

The infrastructure is complete; the focus shifts to data quality.

### Immediate Next Steps (Week 12-13):

- Real Labels:** Integrate SmartBugs and SWC Registry data.
- Enhanced Features:** Add static analysis features (via `slither`) and graph metrics.
- Retraining:** Train models on real data to improve F1-score.

**Final Deliverables (Week 14-15):** We will submit the Final Report (4 pages, full analysis), Presentation Slides, and the documented Code Repository.

## VII. CONCLUSION

We have built a functional ML pipeline for smart contract vulnerability detection, processing real Ethereum contracts and extracting 71 features. While current results are bound by synthetic labels, the system is robust. The final phase involves training on real vulnerability data to achieve the target accuracy.

## REFERENCES

- [1] J. Feist, G. Grieco, and A. Groce, “Slither: A static analysis framework for smart contracts,” in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, May 2019. [Online]. Available: <http://dx.doi.org/10.1109/WETSEB.2019.00008>

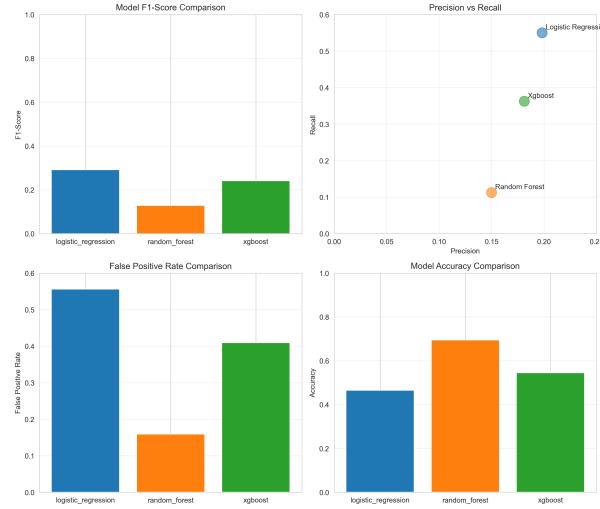


Fig. 2. F1-score and false positive rate comparison across models.

- [2] B. Mueller, “Mythril: Reversing and bug hunting framework for the ethereum blockchain,” <https://github.com/ConsenSys/mythril>, 2018.
- [3] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 254–269. [Online]. Available: <https://doi.org/10.1145/2976749.2978309>
- [4] P. Tsankov, A. Dan, D. D. Cohen, A. Gervais, F. Buenzli, and M. Vechev, “Security: Practical security analysis of smart contracts,” 2018. [Online]. Available: <https://arxiv.org/abs/1806.01143>
- [5] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, “Smart contract vulnerability detection using graph neural network,” in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2020, pp. 3283–3290, main track. [Online]. Available: <https://doi.org/10.24963/ijcai.2020/454>
- [6] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, “Combining graph neural networks with expert knowledge for smart contract vulnerability detection,” *IEEE Transactions on Knowledge and Data Engineering*, 2021. [Online]. Available: <http://dx.doi.org/10.1109/TKDE.2021.3095196>
- [7] J. Yu, X. Yu, J. Li, H. Sun, and M. Sun, “Smart contract vulnerability detection based on multimodal feature fusion,” in *Advanced Intelligent Computing Technology and Applications: 20th International Conference, ICIC 2024, Tianjin, China, August 5–8, 2024, Proceedings, Part III*. Berlin, Heidelberg: Springer-Verlag, 2024, p. 344–355. [Online]. Available: [https://doi.org/10.1007/978-981-97-5588-2\\_29](https://doi.org/10.1007/978-981-97-5588-2_29)
- [8] W. J. W. Tann, X. J. Chang, E. C. H. Liu, D. Pei, and J. K. Sun, “Towards safer smart contracts: A sequence learning approach to detecting vulnerabilities,” <https://arxiv.org/abs/1811.06632>, 2018.
- [9] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, “Smartbugs: A framework to analyze solidity smart contracts,” <https://github.com/smartsbugs/smartsbugs>, 2021.
- [10] P. Qian, Z. Liu, X. Wang, Y. Yin, and Q. He, “Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode,” in *Proceedings of the ACM Web Conference*, 2023.
- [11] Messi-Q, “Smart-contract-dataset,” <https://github.com/Messi-Q/Smart-Contract-Dataset>, 2025.