LLM-Based Vulnerability Detection in Tokenized Assembly: A Case Study on CWE-457

Taylor Marrion
Beacom College of Computer and Cyber Sciences
Dakota State University
Madison, SD, USA
taylor.marrion@trojans.dsu.edu

***Abstract*—This project explores the feasibility of using large language models (LLMs) for vulnerability detection in compiled binaries without source code access. We focus on detecting instances of CWE-457 (Use of Uninitialized Variable) by training transformer-based models on tokenized x86-64 assembly derived from the Juliet Test Suite. Disassembled functions were normalized into abstracted token sequences and used as input for fine-tuning DistilBERT, MiniLM, and MiniBERT models. Among the models evaluated, MiniLM achieved the best performance after hyperparameter optimization, reaching an F1 Score of 0.8189. These results suggest that pre-trained LLMs can learn useful patterns from tokenized binary representations, providing initial validation for this approach. However, further research is needed to extend beyond the limited scope of this study and to assess resilience against obfuscation, optimization, and broader vulnerability classes.**

***Index Terms*—Cybersecurity, Vulnerability Detection, Binary Analysis, Machine Learning, Transformers, Large Language Models, CWE-457**

I. Introduction

As software systems become increasingly complex and security threats continue to evolve, the demand for scalable, automated vulnerability detection techniques has intensified. Traditional methods such as symbolic execution, static analysis, and manual reverse engineering often require access to source code or involve significant human effort, limiting their practicality for analyzing closed-source or compiled binaries.

The emergence of large language models (LLMs) has opened new possibilities for program analysis by enabling machine learning systems to reason about structured sequences such as source code and natural language. Inspired by their success in natural language understanding and code summarization tasks, we investigate whether LLMs can be adapted to detect security vulnerabilities directly from disassembled binary code.

This work presents a focused case study on the detection of CWE-457 (Use of Uninitialized Variable) vulnerabilities in compiled x86-64 binaries. Using a curated subset of the Juliet Test Suite, we normalize disassembly into abstract token sequences and fine-tune transformer-based LLMs to classify functions as either safe or vulnerable. Three transformer models—DistilBERT, MiniLM, and MiniBERT—are evaluated to assess the viability of this approach under controlled conditions.

The primary goal of this project is not to deliver a production-ready vulnerability scanner, but rather to assess the validity of applying LLMs to tokenized assembly for vulnerability detection. By demonstrating promising results within this constrained scope, we aim to lay the groundwork for future research that can expand the methodology to broader vulnerability classes and more challenging real-world binaries.

II. Background

A. Binary Vulnerability Detection

Binary vulnerability detection involves identifying security flaws directly in compiled executable code without access to the original source. Traditional static analysis tools leverage control flow graphs, data flow analysis, and symbolic execution to reason about program behavior. However, these approaches often struggle with scalability, code obfuscation, or optimization artifacts. Techniques like fuzzing and dynamic analysis improve detection coverage but require execution environments and can miss subtle, state-dependent vulnerabilities. As compiled binaries become more complex and proprietary, scalable methods for vulnerability detection without full source context have become increasingly important.

B. Large Language Models for Code Understanding

Large language models (LLMs), such as BERT and its variants, have demonstrated strong performance in tasks requiring the understanding of structured sequences, including natural language, source code, and other highly structured domains. Recent research has shown that transformer-based models can capture both syntactic and semantic relationships in code, enabling tasks such as bug detection, summarization, and functional analysis [1], [2]. These studies suggest that LLMs can learn important behavioral patterns even when applied to technical domains with limited or specialized vocabularies. Building on this insight, our study investigates whether pre-trained LLMs can generalize to the tokenized disassembly of binary programs, treating normalized x86-64 instruction sequences as a ”language” suitable for downstream vulnerability detection tasks. This hypothesis is evaluated under a highly constrained setting focused specifically on detecting instances of CWE-457 (Use of Uninitialized Variable).

C. Related Work

Pordanesh and Tan [2] explored the use of GPT-4 for binary reverse engineering, highlighting LLMs' strengths in function recognition and their limitations in detailed technical reasoning. Anderson et al. [3] applied machine learning to automate subroutine classification in malware binaries using graph-based features, illustrating early efforts to structure binary analysis for learning algorithms. Taviss et al. [1] proposed Asm2Seq, an assembly code summarization technique that reinforces the idea of treating binaries as structured, learnable data sequences. Udeshi et al. [4] demonstrated how symbolic execution could reverse-engineer mathematical equations from binaries, further supporting structured data extraction. Van Zeeland's work [5] on reverse-engineering state machines from C code highlights the effectiveness of abstract behavioral modeling. The angr framework [6] integrates symbolic execution and control flow analysis but also reveals the challenges of path explosion in deep static analysis. Finally, Ramblr [7] improved static binary reassembly through better control flow graph recovery, a challenge closely related to abstracting executable behavior for learning tasks. Collectively, these studies motivate our exploration of LLMs as a potential alternative or complement to traditional binary analysis techniques.

III. Dataset Preparation

A. Dataset Source

The dataset used for this study is derived from the Juliet Test Suite C/C++ Version 1.3 [8], a widely recognized collection of synthetic test cases designed to evaluate the effectiveness of static and dynamic code analysis tools. Specifically, we utilized the Linux-adapted version provided by Richardson [9], which compiles the test cases into x86-64 ELF binaries suitable for disassembly and analysis.

B. CWE-457 Filtering

To limit project scope and maintain a highly controlled experimental setting, we filtered the dataset to include only samples corresponding to CWE-457 (Use of Uninitialized Variable). This vulnerability category was selected due to its prevalence in real-world software and its relatively localized manifestation within functions, making it an ideal starting point for evaluating our proof-of-concept framework.

C. Disassembly and Normalization

Each binary was disassembled using `objdump`, and the resulting assembly code was parsed to extract instruction sequences. To abstract away low-level details and reduce noise, we applied a normalization procedure that mapped instructions into a small vocabulary of semantic tokens:

- **READ**: Memory reads (e.g., `mov eax, [rbp-0x4]`)
- **WRITE**: Memory writes (e.g., `mov [rbp-0x4], eax`)
- **END**: Function returns (e.g., `ret`)
- **NOOP**: All other instructions

This abstraction preserves critical data flow operations while minimizing the model's exposure to irrelevant variations in register usage, instruction encoding, or optimization artifacts.

D. Dataset Export

After normalization, each sample representing a function was saved as a JSONL (JSON Lines) record containing:

- The list of normalized tokens
- The vulnerability label (`safe` or `vulnerable`)
- Metadata fields such as the binary path and disassembly source

The final dataset consisted of 1,852 samples, balanced between 926 vulnerable and 926 safe functions. This ensured that class imbalance would not artificially inflate performance metrics.

E. Motivation for Preprocessing Choices

Our preprocessing pipeline deliberately restricted the input feature space to emphasize high-level data movement patterns rather than relying on raw opcode diversity or register-specific behavior. We hypothesize that vulnerabilities like CWE-457 correlate more strongly with abstract memory access patterns than with specific instruction mnemonics, making semantic normalization a meaningful reduction in complexity for LLM-based learning.

IV. Model Selection

A. Model Family Considerations

Given the goal of detecting vulnerabilities from tokenized disassembly, we focused on transformer-based large language models (LLMs) that are pre-trained on general-purpose text but capable of adapting to structured sequences. Transformer architectures offer strong inductive biases for sequence modeling through mechanisms such as self-attention, making them suitable for learning data flow patterns even outside natural language.

Rather than training from scratch, we leveraged pre-trained models and fine-tuned them on our task-specific dataset. This transfer learning approach reduces resource requirements and benefits from the general linguistic structures already learned during large-scale pre-training.

B. Selected Models

To study the applicability of LLMs in this constrained binary domain, we selected three BERT-derived models with varying size, architecture, and training objectives:

- **DistilBERT (distilbert-base-uncased)** [10]: A compact, distilled version of BERT designed to maintain 95% of BERT's performance while reducing the number of parameters by approximately 40%. DistilBERT serves as a strong, efficient baseline.
- **MiniLM (microsoft/MiniLM-L12-H384-uncased)** [10]: A lightweight model from Microsoft that uses a deep self-attention distillation strategy. MiniLM retains high performance relative to much larger models, making it attractive for resource-constrained settings such as binary analysis.

- **MiniBERT (google/bert_uncased_L-4_H-256_A-4)** [10]: A minimal BERT variant from Google characterized by a reduced number of transformer layers and hidden size. MiniBERT allows investigation of how aggressively model size can be reduced before task-specific performance significantly degrades.

C. Rationale for Model Diversity

By selecting models of varying capacity and design philosophies, we aim to evaluate how model complexity impacts performance when generalizing to a highly structured but non-natural input domain like tokenized assembly. Compact models like MiniLM and MiniBERT are of particular interest for future deployment scenarios where lightweight vulnerability detection engines would be desirable.

Additionally, restricting initial experiments to pre-trained, publicly available models ensures reproducibility and practical relevance, aligning our study with real-world engineering constraints.

V. Training Procedure

A. Tokenization and Input Formatting

Each normalized disassembly sample was processed into a token sequence composed of abstract instruction labels (e.g., `READ`, `WRITE`, `END`). These tokens were concatenated into whitespace-separated strings, treating the disassembly as a natural language input. We then applied a pre-trained tokenizer corresponding to each selected model (e.g., DistilBERT tokenizer for DistilBERT), using truncation and padding to a fixed maximum sequence length suitable for fine-tuning.

B. Dataset Splitting

To enable reliable evaluation, the full dataset was stratified into training and testing splits using an 80/20 ratio. Stratified sampling preserved the balance between safe and vulnerable samples in both subsets. The training set was shuffled to promote generalization, while the test set remained untouched until final evaluation.

C. Training Configuration

Training was conducted using the Hugging Face `Trainer` API with consistent hyperparameters for all baseline models to ensure fair comparison. Key settings included:

- **Batch Size**: 16 samples per device
- **Number of Epochs**: 4
- **Learning Rate**: 2e-5
- **Weight Decay**: 0.01
- **Evaluation Strategy**: Evaluate performance after each epoch
- **Metric for Best Model**: F1-score

During training, the best model checkpoint according to F1-score on the validation set was automatically saved for later use.

D. Evaluation Metrics

Evaluation focused on four standard classification metrics:

- **Accuracy**: Overall fraction of correctly classified samples
- **Precision**: Fraction of predicted vulnerable samples that were correct
- **Recall**: Fraction of actual vulnerable samples correctly identified
- **F1 Score**: Harmonic mean of precision and recall, emphasizing balance

Due to the equal distribution of safe and vulnerable samples, accuracy was informative but F1-score was prioritized as the primary metric for model selection and comparison.

VI. Baseline Evaluation

A. Evaluation Setup

Each selected model—DistilBERT, MiniLM, and MiniBERT—was fine-tuned on the tokenized CWE-457 dataset using identical training hyperparameters. After each epoch, model performance was evaluated on the held-out test set. We report four key metrics: Accuracy, Precision, Recall, and F1 Score, with F1 Score serving as the primary measure for model comparison due to its balanced emphasis on false positives and false negatives.

B. Performance Comparison

Table I summarizes the evaluation results for each model after four training epochs:

TABLE I
Baseline Model Evaluation on CWE-457

Model	Accuracy	Precision	Recall	F1 Score
DistilBERT	0.6927	0.6682	0.7622	0.7121
MiniLM (Microsoft)	0.7709	0.7577	0.7946	**0.7757**
MiniBERT (Google)	0.6011	0.6135	0.5405	0.5747

C. Observations

Among the three models tested, MiniLM demonstrated the highest F1 Score, indicating the best overall balance between detecting vulnerable samples and minimizing false positives. DistilBERT achieved moderate success but lagged behind MiniLM. MiniBERT, while compact, struggled to generalize from the disassembly data, achieving lower accuracy and F1 scores.

These results suggest that model size and representational capacity play critical roles even in this simplified setting. MiniLM's larger hidden size and intermediate depth appear to offer a strong advantage over smaller, more aggressively compressed architectures like MiniBERT.

D. Discussion

Importantly, these findings validate the general feasibility of applying pre-trained LLMs to vulnerability detection from tokenized disassembly, at least within the constrained environment of CWE-457 functions. The gap in performance between

models also highlights the need for careful selection and tuning of model architectures when adapting LLMs to binary analysis tasks.

VII. Hyperparameter Tuning

A. Motivation

Following the baseline evaluation, MiniLM emerged as the most promising model based on F1 Score performance. To further optimize MiniLM's ability to detect vulnerabilities from tokenized disassembly, a focused hyperparameter grid search was conducted. The goal was to determine whether improved training configurations could yield additional gains without modifying the model architecture.

B. Hyperparameter Definitions

Each hyperparameter explored in the grid search serves a critical role in shaping model training, following established best practices in deep learning [11]:

- **Learning Rate**: Controls the step size at each iteration during optimization. A smaller learning rate can lead to more stable convergence, while a larger one speeds up training but risks overshooting minima.
- **Batch Size**: The number of samples processed before updating the model weights. Larger batches provide more stable gradient estimates but require more memory and can lead to poorer generalization.
- **Weight Decay**: A form of regularization that penalizes large weights during optimization to help prevent overfitting.
- **Number of Epochs**: The number of complete passes over the training dataset. More epochs allow the model to learn more thoroughly but can risk overfitting if set too high.

C. Search Space

The hyperparameter tuning process explored combinations of key training parameters known to significantly influence transformer fine-tuning performance:

- **Learning Rate**: {5e-5, 3e-5, 2e-5, 1e-5}
- **Batch Size**: {16, 32}
- **Weight Decay**: {0.0, 0.01}
- **Number of Epochs**: {3, 4, 5}

This yielded a total of 48 hyperparameter combinations tested in an exhaustive grid search.

D. Methodology

For each combination, MiniLM was reloaded from scratch and fine-tuned on the same training and evaluation splits. Evaluation metrics were recorded after each run, with particular focus on F1 Score as the primary selection criterion. The model achieving the highest F1 Score across all runs was selected as the best-performing configuration.

E. Best Configuration

The optimal hyperparameter set identified through this search was:

- **Learning Rate**: 2e-5
- **Batch Size**: 16
- **Weight Decay**: 0.01
- **Number of Epochs**: 5

Using this configuration, MiniLM achieved an F1 Score of 0.8189 on the test set, improving upon the baseline evaluation without requiring architectural changes.

F. Discussion

These results reinforce the importance of fine-tuning even when starting from strong pre-trained models. Small adjustments to learning rate, regularization, and training duration can substantially impact downstream task performance, particularly in domain-shifted tasks like binary disassembly analysis.

VIII. Results and Analysis

A. Baseline Model Performance

The three selected models—DistilBERT, MiniLM, and MiniBERT—were evaluated after initial fine-tuning on the tokenized disassembly dataset. Table II summarizes the best epoch performance achieved by each model in terms of Accuracy, Precision, Recall, and F1 Score.

TABLE II
Baseline Model Evaluation Results

Model	Accuracy	Precision	Recall	F1 Score
DistilBERT	0.6927	0.6682	0.7622	0.7121
MiniLM	0.7709	0.7577	0.7946	0.7757
MiniBERT	0.6011	0.6135	0.5405	0.5747

MiniLM outperformed the other two models across all evaluation metrics, particularly in F1 Score, which served as the primary measure of model quality. DistilBERT showed reasonable performance but lagged behind MiniLM, while MiniBERT underperformed on this task, possibly due to its reduced model size and representational capacity.

B. Effect of Hyperparameter Tuning

Hyperparameter tuning produced notable improvements over the baseline MiniLM configuration. Using the optimized settings (learning rate of 2×10^{-5}, batch size of 16, weight decay of 0.01, and 5 epochs), the final MiniLM model achieved an F1 Score of 0.8189, an increase of approximately 4.3% over its baseline.

TABLE III
Best Hyperparameter Configuration Performance

Accuracy	Precision	Recall	F1 Score
0.8248	0.8448	0.7946	0.8189

The gains demonstrate that careful tuning can significantly enhance performance, even without modifying the model architecture or training data. In particular, increasing the number

of epochs from four to five and applying modest weight decay contributed meaningfully to improved generalization.

C. Error Analysis and Observations

Despite strong overall performance, some misclassifications remained. Preliminary analysis suggests that instances involving subtle instruction ordering or low token diversity (e.g., sequences dominated by repeated 'NOOP' tokens) were more challenging for the model. This highlights the limitations of using heavily normalized disassembly, which removes potentially informative patterns such as specific register use, memory addressing patterns, or instruction arguments.

Nevertheless, the model successfully captured broader behavioral patterns indicative of uninitialized variable use, such as sequences dominated by 'READ' operations without corresponding 'WRITE' instructions beforehand.

D. Summary

The results support the hypothesis that LLMs can generalize vulnerability detection to tokenized binary assembly, at least under constrained conditions. MiniLM proved particularly well-suited to this task, likely benefiting from its compact size and strong representational capacity, allowing it to learn discriminative patterns efficiently from a modestly sized dataset.

These findings motivate further exploration of LLM-based techniques for binary analysis, especially with richer tokenization strategies and expanded vulnerability coverage.

IX. Limitations and Future Work

A. Limitations

While the results of this case study are promising, several important limitations must be acknowledged:

- **Narrow Scope**: The study was restricted to detecting a single vulnerability type (CWE-457: Use of Uninitialized Variable). Broader applicability across other vulnerability classes remains untested.
- **Disassembly Normalization**: By heavily normalizing the assembly code into abstract tokens (e.g., `READ`, `WRITE`, `END`), important contextual information such as register usage, operand values, and precise control flow patterns was lost. This abstraction may limit the model's ability to detect more complex vulnerabilities.
- **Dataset Size and Diversity**: The dataset was relatively small (1,852 samples) and derived from a controlled test suite (Juliet), which lacks the noise, diversity, and obfuscation present in real-world binaries.
- **Lack of Obfuscation and Optimization**: The binaries analyzed were not subjected to compiler optimizations or obfuscation techniques. Real-world binaries often employ such transformations, which could significantly impact model performance.
- **Static Analysis Only**: The approach relied solely on static disassembly, without incorporating dynamic information such as runtime behavior, memory state, or execution traces.

B. Future Work

Building on these initial findings, several promising directions for future research are identified:

- **Multi-CWE Detection**: Extend the dataset to include a broader set of vulnerability types, enabling evaluation of generalization across different semantic patterns of insecure behavior.
- **Enhanced Tokenization**: Explore richer tokenization strategies that retain more structural information, such as control flow tags, operand types, or register dependencies, without overwhelming the model.
- **Obfuscation and Optimization Resistance**: Test the approach on binaries compiled with varying levels of optimization (e.g., `-O2`, `-O3`) and those subjected to obfuscation tools to assess robustness.
- **Hybrid Static and Dynamic Features**: Incorporate dynamic execution features such as instruction traces or memory snapshots alongside static disassembly to provide the model with richer context.
- **Alternative Architectures**: Investigate the use of models specifically pre-trained on code or binary data, such as CodeBERT, GraphCodeBERT, or custom embeddings derived from binary corpora.
- **Scaling to Larger Models and Datasets**: Evaluate whether larger pre-trained models (e.g., DeBERTa, RoBERTa) or transformer variants can further improve performance when sufficient data is available.

These directions offer the potential to validate and extend the approach presented here, with the goal of developing LLM-based binary analysis tools that are robust, scalable, and effective across real-world vulnerability detection scenarios.

X. Conclusion

This case study explored the feasibility of applying large language models (LLMs) to the task of binary vulnerability detection. By treating normalized x86-64 assembly disassembly as a structured input sequence, we investigated whether pre-trained transformer models could effectively learn patterns indicative of vulnerable code without requiring source-level information.

Using the Juliet Test Suite and focusing specifically on CWE-457 (Use of Uninitialized Variable), we conducted baseline evaluations across three compact transformer models and subsequently fine-tuned the best-performing model (MiniLM) through targeted hyperparameter tuning. Our best configuration achieved an F1 Score of 0.8189 on the validation set, demonstrating that LLM-based approaches can detect vulnerabilities in disassembled code with promising accuracy under controlled conditions.

However, it is important to emphasize that this study serves as a proof of concept rather than a production-ready system. The scope was intentionally narrow, the dataset was controlled, and significant challenges remain before such approaches could generalize to real-world binaries.

Despite these limitations, the findings suggest that LLMs have potential for binary vulnerability detection tasks. Future

work expanding to multiple vulnerability classes, enhancing input representations, and testing robustness against real-world complexities will be critical in evaluating the true viability of this approach. Overall, the results warrant cautious optimism and highlight a novel intersection of machine learning, reverse engineering, and cybersecurity research.

ACKNOWLEDGMENTS

The author would like to thank the creators and maintainers of the Juliet Test Suite for providing a standardized benchmark for software vulnerability research, as well as Alexander Richardson (arichardson) for his work in adapting the Juliet Test Suite for Unix-like environments, which greatly streamlined data preparation and automation for this project. Additional thanks go to the Hugging Face community for openly providing the pre-trained transformer models and training tools used in this study. Finally, the related research on applying large language models to code understanding and binary analysis provided valuable inspiration and context for this work.

REFERENCES

[1] S. Taviss, S. H. H. Ding, M. Zulkernine, P. Charland, and S. Acharya, "Asm2seq: Explainable assembly code functional summary generation for reverse engineering and vulnerability analysis," *Digital Threats*, vol. 5, no. 1, Mar. 2024. [Online]. Available: https://doi.org/10.1145/3592623

[2] S. Pordanesh and B. Tan, "Exploring the efficacy of large language models (gpt-4) in binary reverse engineering," 2024. [Online]. Available: https://arxiv.org/abs/2406.06637

[3] B. Anderson, C. Storlie, M. Yates, and A. McPhall, "Automating reverse engineering with machine learning techniques," in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, ser. AISec '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 103–112. [Online]. Available: https://doi.org/10.1145/2666652.2666665

[4] M. Udeshi, P. Krishnamurthy, H. Pearce, R. Karri, and F. Khorrami, "Remaqe: Reverse engineering math equations from executables," *ACM Trans. Cyber-Phys. Syst.*, vol. 8, no. 4, Nov. 2024. [Online]. Available: https://doi.org/10.1145/3699674

[5] D. van Zeeland, "Reverse-engineering state machine diagrams from legacy c-code," Master's Thesis, Eindhoven University of Technology, 2009. [Online]. Available: https://research.tue.nl/en/studentTheses/reverse-engineering-state-machine-diagrams-from-legacy-c-code

[6] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna, "Sok: (state of) the art of war: Offensive techniques in binary analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 138–157. [Online]. Available: https://ieeexplore.ieee.org/document/7546500

[7] R. Wang, Y. Shoshitaishvili, A. Bianchi, A. Machiry, J. Grosen, P. Grosen, C. Kruegel, and G. Vigna, "Ramblr: Making reassembly great again," in *2017 NDSS*, 01 2017. [Online]. Available: https://www.researchgate.net/publication/316913243$_{R}amblr_{M}aking_{R}eassembly_{G}reat_{A}gain$

[8] N. I. of Standards and Technology, "Juliet test suite," https://samate.nist.gov/SARD/test-suites/112, 2024, accessed: 2025-04-25.

[9] A. Richardson, "Juliet test suite c (linux port)," https://github.com/arichardson/juliet-test-suite-c, 2024, accessed: 2025-04-25.

[10] H. Face, "Hugging face transformers models," https://huggingface.co/models, 2024, accessed: 2025-04-25.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org/