EVALUATING THE EFFICACY OF MULTI-TASK ATTENTION NETWORKS IN DETECTING VULNERABILITIES IN WASM BYTE CODE

Sebastian Just
College of Computing & Informatics
Drexel University
Philadelphia, PA 19104
sj992@drexel.edu

Greg Morgan
College of Computing & Informatics
Drexel University
Philadelphia, PA 19104
gm655@drexel.edu

Frank Ottey
College of Computing & Informatics
Drexel University
Philadelphia, PA 19104
fmo28@drexel.edu

Trevor Pawlewicz
College of Computing & Informatics
Drexel University
Philadelphia, PA 19104
tmp365@drexel.com

June 14, 2023

ABSTRACT

The rise of ChatGPT and the power of large-language models, driven by the neural transformer architecture defined in the seminal paper by Vaswani et al. (Attention is All You Need) has garnered significant attention. While media focus has mainly been on the performance resulting from the application of this architecture to large language models in the natural language / text domain, the transformer architecture has proven effective in across a multitude of domains. The general attention mechanisms inherent in these models lend themselves well to solving problems in various domains through the encoding and decoding of domain-specific sequences. Our paper aims to extend the domain of recent advances in LLMs to computer byte code sequences via a fine-tuning process.

Our aim is to investigate the effectiveness of these fine-tuned, large-language, multi-task attention networks for detecting vulnerabilities in WASM byte code artifacts. Building on OpenAI's specialization of the "Whisper" model, which uses attention networks on audio sequences, we begin by focusing on specializing attention networks within large language models specifically for byte code sequences. Shifting from a generative to a discriminator model, we analyze and classify byte code sequences using labeled data. Our study involves fine-tuning existing large language models on WASM byte code sequences, generated by compiling vulnerable and non-vulnerable programs from multiple languages into a common WASM byte code format. This approach allows the network to learn underlying vulnerability patterns within byte code sequences, rather than language-specific sequence patterns. By producing properly labeled vulnerable and non-vulnerable byte code sequence data, we can ensure the integrity and quality of the training data.

To evaluate our approach, we intend to conduct comprehensive experiments on a benchmark dataset that we will create. We will then compare the performance of our model against existing vulnerability detection techniques and tools, using key metrics such as precision, recall, and F1-score to measure the quality of the model in identifying vulnerabilities. The goal will be to ensure that the benchmark dataset covers various common types of vulnerabilities encountered in untrusted code execution, such as memory corruption, code injection, and privilege escalation.

Our preliminary results are expected to demonstrate the potential of our approach in effectively identifying vulnerabilities in untrusted pre-compiled byte code artifacts intended for execution in unknown environments. This research advances automated vulnerability detection techniques, enhancing software security and enabling proactive mitigation strategies. Potential stakeholders interested in this research include the software security industry, web browser engine authors, and virtual machine authors. Further extensions of this research could include automated, byte code level program repair, useful for when access to source code is limited, automated bug detection, or even AI-driven byte code optimization.

Keywords Language models · Byte code analysis · Software security · WASM (WebAssembly)

1 Data and Dataset Selection

Our aim is to extend the recent advancements demonstrated by highly effective multi-task attention network architectures in language modeling to the application of byte code analysis. Specifically WebAssembly (WASM) byte code analysis. To achieve this, a benchmark corpus of labeled training data needs to be generated.

WASM was selected as a suitable byte code candidate due to its significance as a next-generation, language-agnostic, client-side virtual machine target for program execution. Given its role in executing untrusted code on consumer-owned devices, the ability to automatically analyze and detect vulnerabilities in untrusted programs before execution on client hardware holds immense value in the domain of software security. It is crucial to generate byte code sequences from a diverse range of higher-level programming languages that compile down to WASM byte code, as this will provide a valuable dataset for analysis.

In addition to ensuring that byte code sequence examples are collected from a variety of programming languages, we commit to labeling the resulting byte code sequences regarding their disposition as either vulnerable or non-vulnerable code. We will aim to ensure that several classes of vulnerability are represented amongst the training data as well.

2 Model Architecture and Fine-tuning

The excerpt highlights the interest in exploring whether the proposed model can capture semantic similarity of byte code sequences and represent them as vector embeddings in a shared vector space. It draws inspiration from previous work, specifically OpenAI's "Whisper" model, where the transformer architecture was successfully applied to audio sequences.

The machine learning algorithm for this research would involve adapting the transformer architecture to the domain of byte code analysis. The transformer architecture consists of stacked self-attention and feed-forward layers. Self-attention mechanisms allow the model to attend to different parts of the input sequence, capturing contextual dependencies effectively. The feed-forward layers provide non-linear transformations to further process the representations.

In the case of byte code analysis, the transformer-based model would take byte code sequences as input and encode them into vector embeddings. These vector embeddings would ideally capture semantic information and maintain similarity between related byte code sequences. By training the model on a diverse dataset of labeled vulnerability data, the model learns to discriminate between vulnerable and non-vulnerable code based on the embeddings.

To determine the similarity between programs written in different languages, the model's vector embeddings of common byte code sequences can be compared in the shared vector space. This enables the model to measure the semantic similarity between programs, even if they are written in different languages. The vector embeddings provide a compact representation that encapsulates the relevant information about the byte code sequences and allows for efficient computation of similarity metrics.

The machine learning algorithm leverages the power of the transformer architecture to analyze and represent byte code sequences, enabling the identification of vulnerabilities and the assessment of semantic similarity across different programs and languages.

3 Evaluation Metrics

Some evaluation metrics that we are looking to utilize include precision, recall, and the F1-Score. Precision measures the rate of true positives, for example, in our case, correctly detected vulnerable byte code sequences, among all items classified as positives, which would include byte code sequences classified as vulnerable that actually were not. A higher precision score would indicate the model is good at determining the difference between vulnerable and non-vulnerable byte code sequences.

Recall, on the other hand, also known as sensitivity or true positive rate, measures the proportion of true positives that are correctly detected among all actual vulnerabilities present in the dataset. A high recall indicates the ability to identify a significant portion of vulnerabilities, minimizing false negatives (missed vulnerabilities).

F1-Score: The F1-score is the harmonic mean of precision and recall, providing a balanced assessment of both metrics. It offers an overall measure of the model's performance by considering both false positives and false negatives. A higher F1-score indicates a better trade-off between precision and recall.

4 Comparative Analysis

There has been some prior research in this area, as evidenced by the existing literature in our references section. However, our research introduces several novel elements in our experimentation.

Firstly, our exploration of WebAssembly (WASM) as a byte code target for language modeling is relatively new. WASM has gained popularity due to the growing need to run untrusted code on the web within sandboxed environments. While sandboxing provides some level of security, no environment is completely isolated. By improving the trustworthiness of untrusted code through pre-execution analysis via a language model, we aim to improve overall security.

Another aspect that contributes to the novelty of our research is the use of a dataset comprising bytecode sequences generated from a diverse set of programming languages. Existing studies in this field have primarily focused on specific languages and specific problems. In contrast, our objective is to enable the model to operate more generally on byte code, without being limited to recognizing patterns specific to a particular programming language.

5 Implications and Future Work

By establishing that neural language models built from attention networks can work with byte code sequences as well as they work with other sequences, we might be able to leverage these systems to perform automated security scanning of untrusted byte code streams, or allow us the measure the semantic similarity of different byte code sequences (programs)

One particular area of research that could be interesting is to see if we could build AI systems to use their knowledge of bytecode sequences to perform optimizations on bytecode sequences.

6 Project Plan

In order to meet our objectives, we will need to propose a fairly ambitious timeline.

6.1 Week 1: Project Setup and Literature Review

- Establish the project environment by setting up the required software and tools necessary for conducting the research on vulnerability detection in WASM byte code artifacts, including the installation and configuration of relevant frameworks, libraries, and development environments.
- Conduct an in-depth literature review on vulnerability detection, attention networks, and WASM byte code analysis.
- Refine the research questions and objectives based on the literature review.

6.2 Week 2: Data Collection and Preprocessing

- Identify and gather a diverse set of vulnerable and non-vulnerable WASM byte code artifacts.
- Preprocess the collected data, ensuring proper labeling and formatting for training the models.

6.3 Week 3-5: Language Model Adaptation and Training

- Choose a suitable large language model as the base architecture.
- Modify the model to specialize in byte code sequence analysis instead of natural text.
- Implement the necessary modifications to enable multi-task learning for vulnerability detection.
- Train the adapted model using the labeled WASM byte code sequences.
- Define appropriate loss functions and evaluation metrics for the training process.
- Conduct multiple training iterations, monitoring and recording the model's performance.

6.4 Week 6-7: Model Evaluation and Refinement

- Evaluate the model on a validation dataset to assess its performance.
- Measure key metrics such as precision, recall, and F1-score to determine the accuracy of vulnerability detection.
- Analyze the performance of the model and identify areas for improvement.
- Refine the model further by adjusting hyperparameters, loss functions, or incorporating additional training data if necessary.
- Repeat the evaluation process to measure the impact of the refinements.

6.5 Week 8-9: Results Analysis and Conclusion/Future Works

- Evaluate the performance of the model on the test dataset.
- Compare the model's results against existing vulnerability detection techniques and tools.
- Identify strengths, weaknesses, and potential areas of further improvement for our approach.
- Interpret the findings and discuss the implications of our approach in detecting vulnerabilities in WASM byte code artifacts.
- Based on results, determine potential areas of future work.

6.6 Week 10: Report Writing and Finalization

- Compile all the findings, methodology, and analysis into a comprehensive research paper.
- Write the introduction, methodology, results, discussion, and conclusion sections.
- Revise and proofread the paper.

References

Nami Ashizawa, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura. Eth2vec: Learning contract-wide code representations for vulnerability detection on ethereum smart contracts. In *Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '21, page 47–59, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384001. doi:10.1145/3457337.3457841. URL `https://doi.org/10.1145/3457337.3457841`.

Matthieu Jimenez, Cordy Maxime, Yves Le Traon, and Mike Papadakis. On the impact of tokenizer and parameters on n-gram based code analysis. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 437–448, 2018. doi:10.1109/ICSME.2018.00053.

Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, and Andrea Janes. Big code != big vocabulary: Open-vocabulary models for source code. *CoRR*, abs/2003.07914, 2020. URL `https://arxiv.org/abs/2003.07914`.

Sunghun Kim, Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, and Shivkumar Shivaji. Predicting method crashes with bytecode operations. In *Proceedings of the 6th India Software Engineering Conference*, ISEC '13, page 3–12, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450319874. doi:10.1145/2442754.2442756. URL `https://doi.org/10.1145/2442754.2442756`.

Alexander LeClair, Siyuan Jiang, and Collin McMillan. A neural model for generating natural language summaries of program subroutines. *CoRR*, abs/1902.01954, 2019. URL `http://arxiv.org/abs/1902.01954`.

Ao Pu, Xia Feng, Yuhan Zhang, Xuelin Wan, Jiaxuan Han, and Cheng Huang. Bert-embedding-based jsp webshell detection on bytecode level using xgboost. *Security and Communication Networks*, 2022:4315829, Aug 2022. ISSN 1939-0114. doi:10.1155/2022/4315829. URL `https://doi.org/10.1155/2022/4315829`.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.

Muhammad Fakhrur Rozi, Sangwook Kim, and Seiichi Ozawa. Deep neural networks for malicious javascript detection using bytecode sequences. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. doi:10.1109/IJCNN48605.2020.9207134.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

Huaiguang Wu, Hanjie Dong, Yaqiong He, and Qianheng Duan. Smart contract vulnerability detection based on hybrid attention mechanism model. *Applied Sciences*, 13(2), 2023. ISSN 2076-3417. doi:10.3390/app13020770. URL `https://www.mdpi.com/2076-3417/13/2/770`.