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 by Vaswani et al. [2017b] 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 problem-solving across a multitude of domains [Radford et al., 2022] as the attention mechanisms inherent in these models lend themselves well to solving problems sequence-to-sequence learning. 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 Radford et al. [2022]'s work, which uses attention networks on audio sequences, we begin by focusing on specializing attention networks within large language models specifically for byte code sequences. There exists prior art for this attempt by Pu et al. [2022], Ashizawa et al. [2021], and Rozi et al. [2020]. Shifting from a generative to a discriminator model, we aim 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. Much of the challenge will be in tokenizing our byte code sequences.

The general applicability of sequence-to-sequence learning as it relates to byte code sequence encoding as been explored as a property of the "natural-ness" of byte code sequences in Choi and Nam [2023]. Here, it is shown that byte code, while more natural than source code, is less natural than AST's. The hope is that byte code sequences are just natural enough that sequence-to-sequence learning mechanisms of attention networks could extend to work on them.

One of the key elements regarding training a neural language model on byte code sequences will be tokenization of these sequences. Jimenez et al. [2018] have shown that in practice, when utilizing language models on source code itself, the tokenization process significantly impacts experimental results. Considering that WASM byte code is of variable-length, some experimentation with tokenization may be necessary to produce desirable results.

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 prior to execution remains critical.

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

Our approach leveraging multi-task attention draws inspiration from previous work, one specific example being OpenAI's "Whisper" model [Radford et al., 2022], where the transformer architecture was successfully applied to audio sequences in a multi-task way with mild supervision to build a model that performed well across various automatic speech recognition, ASR tasks. This research shows that transformer architectures generalize well in various sequence-to-sequence learning settings.

Indeed, the application of neural machine architectures to the program analysis domain can also be found in Pu et al. [2022] and Ashizawa et al. [2021]. Both similarly take inspiration from the neural language modeling domain and attempt to apply it to software security. In our case, the novel contribution would be application of neural architectures as applied to WASM byte code. One of the key differences between that may be worth studying is the difference an auto-regressive approach, like the one taken by GPT models, vs the bi-directional encoder-decoder approach used in models like BERT[Devlin et al., 2019].

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; this property was also explored by Ashizawa et al. [2021] to determine if there vulnerabilities in the smart contracts represented in Etherium byte code sequences. By training our model on labeled vulnerability data from WASM byte code sequences, we could further extend Ashizawa et al. [2021]'s work and apply it to discriminate between vulnerable and non-vulnerable code based on the embeddings of those byte code sequences.

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, and thus missed vulnerabilities. Other metrics that are commonly used in these types of analysis are the F1 score and the ROC score which both give a more comprehensive view of the performance of the model in question vs. either the precision or recall alone.

4 Comparative Analysis

There is a promising amount of prior literature in the area that indicates that the approach we aim to take is sound. Of the references, Rozi et al. [2020], Pu et al. [2022], and Ashizawa et al. [2021] are the most relevant prior art. Rozi et al. [2020]'s work in particular is relevant regarding the application of a similar technique to running untrusted code within the context of V8 Javascript engine byte code sequences. That said, while our work is quite similar in some ways, in other ways it is novel.

Firstly, our exploration of WebAssembly (WASM) as a byte code target for language modeling is relatively unique as compared to the above three works. While WASM has gained popularity as a method to run untrusted code on the web within sandboxed environments, 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. In addition, unlike the byte code sequences used in the above work, WASM byte code is specifically designed for as a language-agnostic compile-time target. This means it has more generic structures than the other three byte code targets.

Another aspect that contributes to the novelty of our research is the use state-of-the-art language modeling techniques. Rozi et al. [2020]'s work is most similar in the problem domain, but uses more traditional techniques such as CNNs, LSTMs, and RNNS vs. more modern attention mechanisms. Pu et al. [2022] on the other hand uses attention mechanisms from a pre-trained BERT model [Devlin et al., 2019], but does not build a multi-task, mild supervision system like the one defined by Radford et al. [2022]. Finally, the work by Ashizawa et al. [2021] also eschews the attention mechanism of transformers and instead relies on adaptations of word2vec [Mikolov et al., 2013] called PV-DM [Le and Mikolov, 2014] for handling the document case. What's interesting is that Ashizawa et al. [2021]'s work is late enough such that Vaswani et al. [2017b]'s work could have been explored as a basis for exploration, but the work doesn't seem mentioned.

5 Implications and Future Work

By establishing that multi-task attention models trained with byte code sequences can generalize towards byte code level program understanding, we would be able to build automated systems of general program analysis that could safely analyze untrusted code without necessarily requiring the original source code or AST transformations of that code. Such systems would be a major boon to the software security landscape which could use these as components in threat modeling and threat detection systems. Other powerful applications could be analyzing program fragments to see how similar they are to other programs, or automated program repair, by having the attention model attempt to "fill-in" the "next token" or byte code sequence in a damaged program. Finally, one particular area of research that could be interesting is to see if a system with bytecode sequence understanding could perform *optimizations* on bytecode sequences, dynamically re-writing them. Such an application could be useful in either an AOT byte code compiler or a JIT.

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`.

Yoon-Ho Choi and Jaechang Nam. On the naturalness of bytecode instructions. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394758. doi:10.1145/3551349.3559559. URL `https://doi.org/10.1145/3551349.3559559`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

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`.

Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014. URL `http://arxiv.org/abs/1405.4053`.

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`.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

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, 2017a. URL `http://arxiv.org/abs/1706.03762`.

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, 2017b. 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`.