Data augmentation to break WebAssembly classifiers

1st Javier Cabrera Arteaga
PhD. student at SCS, KTH Royal Institute of Tecnology
Stockholm
FID3214 course final report
javierca@kth.se

***Abstract*—In this work we proposed a data augmentation technique using a novel mutation tool for WebAssembly that provides semantically equivalent code transformations. We reproduce MINOS, a novel tool to automatically detect malicious WebAssembly programs. We empirically demonstrate that the original dataset of MINOS is too small to be generalized. The MINOS model trained with the original dataset dropped its accuracy to 50% on the augmented datasets. Besides, the model does not improve when it uses the augmented dataset for training. Thus, we show that the proposed normalization process in MINOS is affected under depth data transformations for WebAssembly.**

***Index Terms*—Data augmentation, WebAssembly, Automatic classification, Crypto malware**

I. Introduction

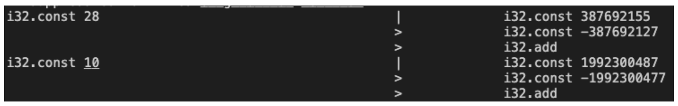
As soon as WebAssembly was supported by major browsers [1], [2], the presence of malwares was reported [3]. For an instance, due to its performance, an undesired cryptominer code might be executed in the client browsers, using the computing power of million of potential users without their consent. Thus, the detection of unwelcome Wasm binaries is important. Recent works proposed several ways to automatically detect malicious Wasm binaries, from static analysis of the binaries to dynamic detection at runtime [4]–[6].

Naseem et al. [7] suggested a novel method, in terms of performance and accuracy, to automatically detect Wasm malwares. They use a dataset of 300 binaries, 150 benign and 150 malign WebAssembly programs, to create a convolutional neural network classifier. We have observed that such dataset is too small to be generalized and the validation process on obfuscation is not enough.

Previous works used data augmentation through semantically equivalent code transformation, to break and enhance automatic classification tools [8], [9]. To use semantically equivalent transformation for code helps to tackle the fact that malware often uses obfuscation to escape detection mechanisms. In this work, we empirically demonstrate that this data augmentation technique bypasses the MINOS baseline classifier. In addition, we evaluate if the same data augmentation technique enhances the MINOS baseline classifier.

We use wasm-mutate to augment the initial dataset proposed from MINOS. wasm-mutate[1] was recently released as a new tool for fuzzing Wasm compilers, runtimes, validators, and other Wasm-consuming programs. wasm-mutate has the ability to apply semantics-preserving changes to the input Wasm module. A variant or mutant from wasm-mutate computes identical results when given the same inputs as the original Wasm module. wasm-mutate works at the code instruction level, taking one Wasm binary as input and providing a new variant from it. Thus, it can be used to augment the MINOS dataset while preserving the classification labels in the augmented dataset.

In Figure 1 we illustrate a mutation example. wasm-mutate has a virtually infinite space of code transformations. The example in Figure 1, takes a static constant from a random place in the code and split it in the sum of two random numbers, this sum resulted in the original constant during the program execution. Only this transformation offers literally an infinite possibility of statically different programs.



```
i32.const 28                                  |           i32.const 387692155
                                              >           i32.const -387692127
                                              >           i32.add
i32.const 10                                  |           i32.const 1992300487
                                              >           i32.const -1992300477
                                              >           i32.add
```

Fig. 1: Wasm textual format for a constant unfold mutation, left side the original constant instruction, right side its mutation.

We formulate the research questions that we aim to investigate through our experimentation:

- How effective is the MINOS model trained with the original dataset in classifying semantically equivalent variants generated by wasm-mutate?
- Can a MINOS-based classifiers be enhanced by adversarial training with wasm-mutate augmented data?

II. Method

In this section, we state our methodology to answer our research questions. Besides, we describe how we build the datasets and the experimental setup.

[1]https://github.com/bytecodealliance/wasm-tools/tree/main/crates/wasm-mutate

A. *Datasets*

We answer our research questions by using three datasets: the original dataset proposed by MINOS and two augmented datasets. We built the latter two using the original one, i.e. all generated binaries are semantically equivalent variants of the binaries in the original dataset. The difference between the two augmented datasets is the random seed used to perform the code transformations. This setup will help us to answer our second RQ. Our intuition is that even when an augmented dataset is used to train the model, the model will fail on another augmented dataset because the code transformation space is virtually infinite.

The binaries in the dataset suggested by Naseem et al. [7] are not provided and we can only find the source website and how to collect the binaries. We have partially collected the proposed dataset. We lack the binaries reported from VirusTotal and VirusShare [10]. The main reason is that collecting binaries using VirusTotal and VirusShare requires a long term setup[2]. Besides, according to Hilbig et.al. [11], the prevalence of cryptominers have decreased or the sites containing the malwares are not available. On the other hand, Hilbig et al. mentioned that the majority of the malwares are redundant, i.e. several malware binaries are the same and only the source website is different.

The main issue with the previous analysis is that the original dataset becomes imbalanced, concretely, 134 BENIGN programs and 35 MALIGN programs, for a ratio of 1 malware out of 4 programs. To tackle a possible misleading of the results, we use SMOTE [12] to complete the original dataset. SMOTE is an oversampling method that works by creating synthetic instances from the minor class.

In Table I we listed the dataset specifications. The table is composed of three columns: name of the dataset, number of benign binaries and the number of malign binaries. Each row corresponds to a dataset used in this work: ORIGINAL, AUGMENTED1 and AUGMENTED2. The binaries of ORIGINAL have the `MALIGN` or `BENIGN` labels from the previous work of MINOS.

Name	Benign	Malign
ORIGINAL [3]	130	130
AUGMENTED1	20000	20000
AUGMENTED2	10000	10000

TABLE I: WebAssembly datasets

[2]One easy way to collect the full dataset could be to query each binary in the Wasmbench collection in VirusTotal. However, a public subscription only allows 500 requests per day, making possible to check all the 8461 binaries in 16 days

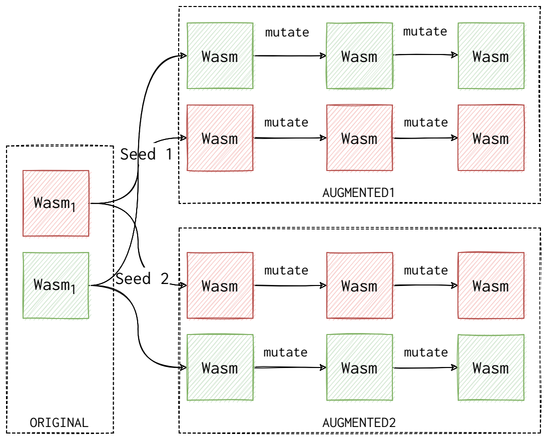


Fig. 2: Generation process for the augmented datasets.

We follow the process illustrated in Figure 2 to build the two augmented datasets. We pass each binary in ORIGINAL to wasm-mutate in an iterative process, generating up to 1k mutants for a single binary. The output variant from a previous iteration is the input for the next iteration, as it is showed with the arrows in the figure. We repeat this process twice with two different random seeds, generating AUGMENTED1 and AUGMENTED2. Notice that, the instances in the two augmented datasets contain intrinsically the right classification.

B. *MINOS reproduction.*

We reproduced the algorithm proposed by MINOS for classification. We validate its proposal, and then we evaluate the two augmented datasets to observe the impact of our data augmentation technique.

Naseem et al. proposed a novel and fast normalization process in MINOS. One single Wasm binary can be small as a couple of bytes or huge as the maximum 2Gb allowed in the browser. Their approach proposes to create a square image from each Wasm binary using its bytes as pixel's intensity and reshaping the resulting image to 100×100 pixels. We generate the grayscale images from the WebAssembly binaries in all datasets. Thus, each image can be seen as a vector of $10000 = 100 \times 100$ features. In Figure 3, we illustrate two binaries in their grayscale image representation.

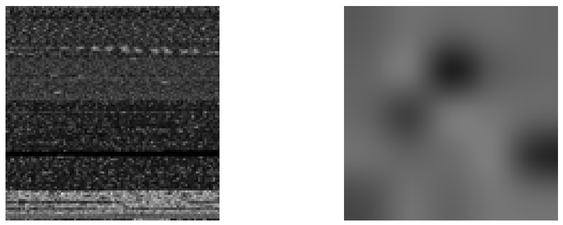


Fig. 3: Grayscale representation of two Wasm binaries

We create a Convolutional Neural Network (CNN) following the specification of the authors, three sequential convolution layers of 16, 32 and 64 filters respectively with a 3×3 kernel size interleaved with max-pool layers of size (2,2), followed all by a dense layer. The latter returns two possible values, for `MALIGN` or `BENIGN` labels. We pass the dataset to the model for training in 50 epochs. We run our experiments in a Standard NC6 machine (6 vcpus, 56 GiB memory) on Azure with a 12Gb memory GPU. We made the notebook of the implementation, as well as the datasets, publicly available for the sake of open-science and reproducibility. The implementation and the datasets could be found at https://github.com/Jacarte/ralph and, https://zenodo.org/record/5832621#.YdwzDCx7kUE respectively.

C. Evaluation

The first step to answer our research questions is to validate the results proposed in MINOS [7]. To do so, we partially reproduce their workflow and metrics. We divided ORIGINAL in training and testing sets using an 80-20 configuration. Next we pass AUGMENTED1 to the model trained with ORIGINAL, we then collect the metrics proposed by MINOS to measure the impact of our data augmentation technique. The final and third step is to test AUGMENTED2 on a model trained on AUGMENTED1 to see the impact of a different distribution in a model trained with an augmented data.

We use Receiver Operating Characteristic(ROC) analysis and the Accuracy metrics to evaluate the performance of the classifier at every step. The ROC analysis will show how well the instances of the `MALIGN` class can be separated from the `BENIGN` class in each dataset, while the Accuracy hints on the actual performance of the classifier. In addition, we analyzed the accuracy and the loss during every training process of the CNN model.

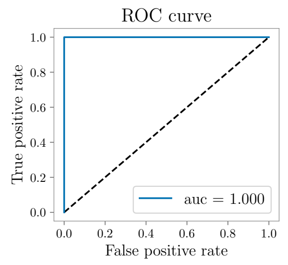
III. Results

A. *How effective is the MINOS model trained with the original dataset in classifying semantically equivalent variants generated by wasm-mutate?*

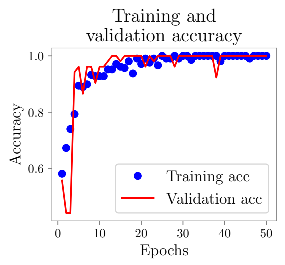
We reproduced the algorithm proposed in MINOS [7] and we trained a CNN with the original dataset. In Figure 4 we illustrate, from left to right, the ROC curve, the accuracy of the model per epoch and learning curve (the loss per epoch). The model behaves as reported in the MINOS work, having an AUC value of 1.0 and an Accuracy of 0.97 by using the ORIGINAL dataset. Besides, the learning curves are similar.

In Figure 4a we can observe an area under the curve of 1. This indicates that the model's can distinguish between the two classes, `MALIGN` and `BENIGN` training a model with ORIGINAL. The model converges to 100% accuracy on the training and the test sets before reaching the maximum number of epochs, as Figure 4b shows. The learning curve in Figure 4c that training and testing loss decreases and stabilizes. Thus, on the ORIGINAL dataset, the model generalizes effectively.

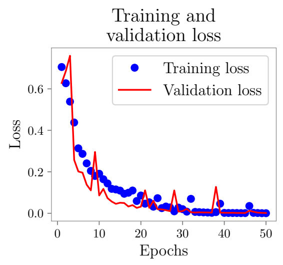
To evaluate the impact of the augmented data provided by wasm-mutate we test AUGMENTED1 on a model trained with ORIGINGAL. As Figure 5 illustrates, the model cannot distinguish between the two classes, having an AUC value of 0.397. The accuracy of the model is 0.5, the same as flipping a coin.



(a) ROC curve for a model trained with ORIGINAL.



(b) ORIGINAL dataset and model training accuracy per epoch



(c) ORIGINAL dataset and model training loss per epoch

Fig. 4: MINOS proposal on the ORIGINAL dataset.

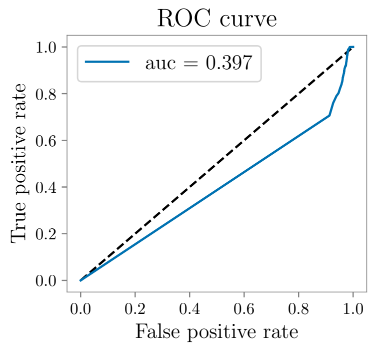
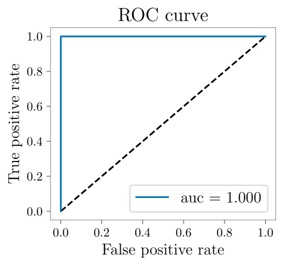


Fig. 5: Resulting ROC curve after testing AUGMENTED1 in a model trained on ORIGINAL.

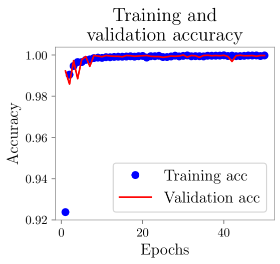
MINOS approach fails to detect the variants generated with wasm-mutate. In their work, Naseem et al. proposed an experiment to evaluate their model againts obfuscation. However, the obfuscation method proposed only affects a small piece of the Wasm binaries, they only tried by changing the name of the functions inside the programs. wasm-mutate on the other hand, offers a smarter and massive way to augment Wasm datasets through semantically equivalent transformations of code. Therefore, program variants generated with wasm-mutate can bypass a MINOS classifier trained with WebAssembly programs in the wild.

B. Can a MINOS-based classifiers be enhanced by adversarial training with wasm-mutate augmented data?

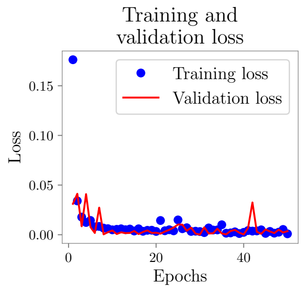
We trained a model on AUGMENTED1 and the area under the ROC curve is 1, as Figure 6a illustrates. The model becomes accurate, and the learning curve shows that the model also generalizes on AUGMENTED1, as it can be appreciated in Figure 6b and Figure 6c respectively.



(a) ROC curve for a model trained with AUGMENTED1.



(b) Model training accuracy per epoch for AUGMENTED1.



(c) Model training loss per epoch for AUGMENTED1.

Fig. 6: MINOS proposal trained on the AUGMENTED1 dataset.

We also tested the model trained with AUGMENTED1 by passing AUGMENTED2 to it. The accuracy of the model is approx. 0.55 and the ROC curve can be appreciated in Figure 7 with an area under the curve of 0.586. The model is barely able to distinguish between the classes. We generated two augmented datasets by following two different random seeds. Training with one makes the classifier to fail on the other. As we stated, the transformation space that wasm-mutate offers is virtually infinite, thus, this same pattern can be always be followed, given a dataset for training, another completely different can be constructed using wasm-mutate, making the classifier to fail.

On the other hand, one might think that the model performance can be improved by training with a larger dataset. We collected the reward on performance when we train a model with larger datasets, having AUGMENTED2 for testing. We have observed that the reward by having AUGMENTED1 more than double sized is not remarkable and that is diminishing while the training dataset size increases [4].

[4]A deeper analysis can be found at Appendix A

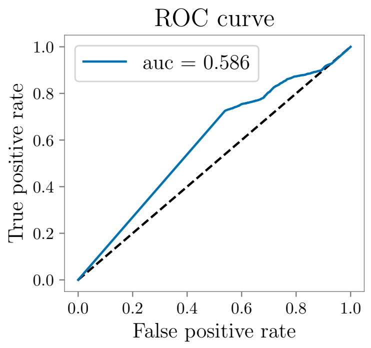


Fig. 7: ROC curve for testing AUGMENTED2 in a model trained with AUGMENTED1.

IV. Conclusions

In this work, we evaluate the performance of the approach proposed by Naseem et al. in MINOS [7]. We demonstrated that the data augmentation provided by wasm-mutate breaks a MINOS classifier trained only with the WebAssembly binaries in the wild. wasm-mutate is the first tool able to augment Wasm datasets using semantically equivalent transformations. However, the model proposed in MINOS cannot be improved by training it with the outcome of wasm-mutate. We would like to remark that placing a defense to the WebAssembly level is not necessarily the best option. Wasm is an intermediate language, and we have observed that the majority of the code transformations by wasm-mutate are removed during the underlying compilation of the browser. Thus, to have wasm-mutate to obfuscate code is perfectly feasible without adding overhead to the WebAssembly that is actually executed.

We found several more ways than MINOS for malware detection, therefore, the reproduction of MINOS is only our first step. In future works, we will evaluate runtime based classifiers like SEISMIC [5] with the technique proposed in this work. Nevertheless, wasm-mutate can be used to canonize WebAssembly binaries, removing obfuscation on the fly. A mixing between wasm-mutate and some automatic classifiers can be constructed to harden the task of detecting undesired WebAssembly programs.

References

[1] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, “Bringing the web up to speed with WebAssembly,” in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017, pp. 185–200.

[2] “WebAssembly: The 4th Official Language of the Web.” [Online]. Available: https://www.cognizantsoftvision.com/blog/webassembly-the-4th-official-language-of-the-web/

[3] M. Musch, C. Wressnegger, M. Johns, and K. Rieck, *New Kid on the Web: A Study on the Prevalence of WebAssembly in the Wild*, 06 2019, pp. 23–42.

[4] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna, “Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense,” 10 2018, pp. 1714–1730.

[5] W. Wang, B. Ferrell, X. Xu, K. W. Hamlen, and S. Hao, "SEISMIC: SEcure in-lined script monitors for interrupting cryptojacks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11099 LNCS. Springer Verlag, 2018, pp. 122–142.

[6] A. Romano and W. Wang, "WASim: Understanding WebAssembly Applications through Classification," *Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*, pp. 1321–1325, 2020.

[7] F. Naseem, A. Aris, L. Babun, S. Uluagac, and E. Tekiner, "MINOS: A Lightweight Real-Time Cryptojacking Detection System," *Ndss*, no. February, 2021. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_4C-4_24444_paper.pdf

[8] W. Zhang, S. Guo, H. Zhang, Y. Sui, Y. Xue, and Y. Xu, "Challenging Machine Learning-based Clone Detectors via Semantic-preserving Code Transformations," pp. 1–15, 2021. [Online]. Available: http://arxiv.org/abs/2111.10793

[9] S. Srikant, S. Liu, T. Mitrovska, S. Chang, Q. Fan, G. Zhang, and U.-M. O'reilly, "GENERATING ADVERSARIAL COMPUTER PROGRAMS USING OPTIMIZED OBFUSCATIONS." [Online]. Available: https://github.com/ALFA-group/adversarial-code-generation

[10] "VirusTotal - Home." [Online]. Available: https://www.virustotal.com/gui/home/search

[11] A. Hilbig, D. Lehmann, and M. Pradel, "An empirical study of real-world webassembly binaries: Security, languages, use cases," *Proceedings of the Web Conference 2021*, 2021.

[12] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *arXiv e-prints*, p. arXiv:1106.1813, Jun. 2011.

APPENDIX

In the following we present the results of training the model with larger datasets.

In Figure 8 we illustrate the reward in AUC and Accuracy of the model when we train it with datasets of the AUGMENTED1 distribution. The x axis shows the increasing in the size of the dataset. The y axis shows the difference between the current AUC or accuracy and the previous dataset size. In all cases, the dataset contains the same number of instances for the MALIGN and BENIGN classes. We have observed that the model gets saturated after increasing the dataset by 20000 instances.

We also show the values for the AUC and the Accuracy versus the size of the dataset in Figure 9. The value of the accuracy is always maintained below 0.6 excepting for small datasets, less than 4000 instances. Our intuition is that the accuracy is always above 0.5 and not completely aleatory because the two augmented datasets share instances, after or before the normalization process. This sharing can be a consequence of two main factors, first, the outcome of wasm-mutate can be generating the same variants even when the seeds are different and second, the normalization process creates the same representation for two variants of the same program. In future work, we plan to dig into these issues.

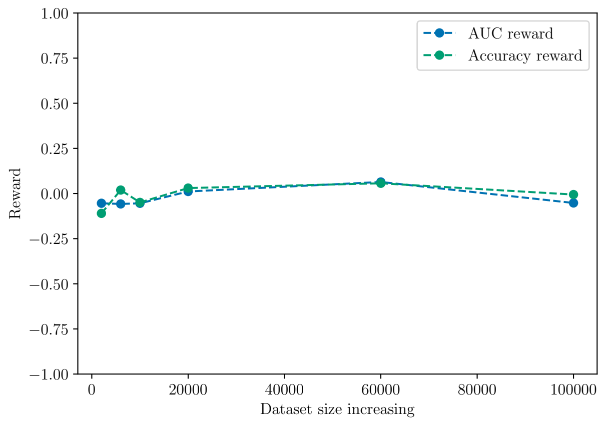


Fig. 8: Reward on AUC and Accuracy of the model training it with different dataset sizes and testing it with AUGMENTED2.

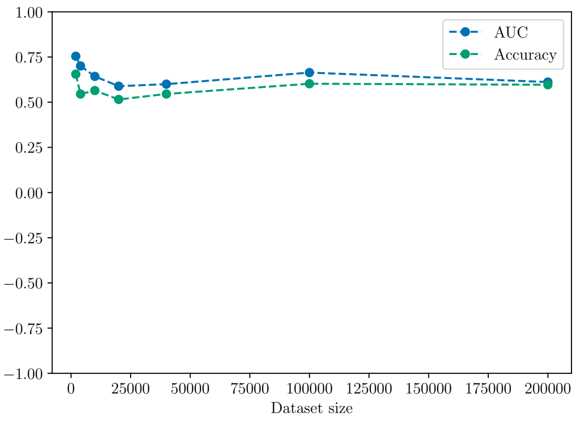


Fig. 9: AUC and Accuracy of the model training it with different dataset sizes and testing it with AUGMENTED2.