Evading Malware Detection with Assembly-Level Code Generation

***Abstract*—**

I. Background

The arms race between malware and the detection of malware is an ongoing struggle. Machine learning (ML) and deep learning (DL) detection systems have been shown to be an effective tool to identify malicious software statically, e.g., the DREBIN [1], MalConv [2] and EMBER [3] detectors. While able to perform well on unseen examples, these models are vulnerable to adversarial evasion attacks, when an adversary perturbs an input such that the classifier fails to correctly identify it [5]. In the case of a practical adversarial malware evasion attack, the adversary modifies the malware binary such that it maintains the correct format, is executable, and preserves its malicious functionality [6]. It is important to discover the ways adversaries with ill intentions can conduct these attacks. To this end, the cybersecurity community has invested a great deal of effort to develop adversarial attacks [6]–[9]. Proposing such attacks allows researchers to devise countermeasures to defend against them [10].

Meanwhile, in the late 2010s, the field of natural language processing (NLP) was revolutionized by two important advances: the Transformer architecture for sequence processing [11] and task agnostic pre-training of large language models (LLMs) like ELMo [12] prior to task-specific fine-tuning. These advances resulted in a series of models that achieved state of the art performance at a variety of NLP tasks, e.g., GPT [13] and BERT [14], along with methods to use multiple LLMs in same model architecture to expand their capabilities [24]. Since source code and natural language contain structural similarities, researchers applied these new strategies to tasks like automatic code documentation, e.g., CodeBERT [15], code generation, e.g., codex [16], and transcompilation [17]. Most of the research related to source code has been done for high-level languages like Python since there is more demand for code tools in these languages.

Recently, these advances have been applied to improve low level source code representations. Finding an effective representation for code at this level has long been a challenge. An effective representation of machine code is useful for a variety of applications, such as binary code similarity detection [18], function type signatures inference [19], value set analysis [20], and malware detection at the byte-level [2], [4]. Previous works used simple raw byte encodings [20] or assembly-level encodings [18], neither of which can provide rich semantic knowledge about the meaning of the instructions. [21] and [22] respectively introduced PalmTree and BinBERT, LLMs for assembly code based on the BERT [14] architecture and unsupervised pre-training paradigm. [23] adopted the methodology from [15] to generate assembly code from natural language descriptions.

II. Proposal

In this work, we propose a novel adversarial evasion attack to evade malware classifiers. Our attack identifies suspicious sections of executable code within the binary and substitutes them with semantically equivalent code that appears harmless. Unlike previous works which proposed a rule-based substitution strategy to modify the opcode distribution of malware, we use a novel sequence to sequence (seq2seq) model to make large portions of malicious code appear benign. Our model uses an encoder-decoder architecture to perform the sequence mapping. We use PalmTree as an encoder and GPT as a decoder. Prior to conjoining the encoder and decoder, each component is pre-trained in an unsupervised fashion on assembly level code.

We include two state of the art detection models when we evaluate the efficacy of our attack: MalConv and EMBER. Since our method only modifies the code of the malware, we develop a third classifier that only examines the source code itself when making its classification. Previous works suggest that malware classifiers likely pay more attention to the header and metadata portions of malware than the source code itself []. Therefore, we argue that our method should be used in conjunction with a different adversarial attack, one that has a mechanism to manipulate the header and metadata of the malware. To test this hypothesis, we introduce a second evasion attack, test it against all three of our classifiers, and demonstrate that the evasion rate of the two attacks combined is higher than the evasion rate of either attack in isolation.

III. Discussion

A. What I am confident about

Previous work suggests that the proposed model architecture could be effective. Unsupervised pre-training of the encoder and decoder should be feasible using training objectives discussed in previous works. Gathering binary source code and dissassembling it to create and unlabeled dataset for pre-training should prove simple. Finally, setting up the experiment involving different malware classifiers and different evasion attacks should also be feasible. Whether or not the results will be positive remains to be seen.

B. What will be challenging

What will be a challenge is determining what makes a block of malware code appear suspicious and what makes it appear benign. Taking these concepts and turning it into a training objective to actually perform the sequence-to-sequence training will also be challenging. Gathering data for this task might also be difficult. To overcome these challenges, I need to read more about seq2seq models, machine translation, and transcompilation, and unsupervised learning methods for performing these tasks. I also need to read more about malware and perhaps explainability to figure out properties of malware vs benignware.

REFERENCES

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[2] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] H. S. Anderson and P. Roth, "Ember: an open dataset for training static pe malware machine learning models," *arXiv preprint arXiv:1804.04637*, 2018.

[4] E. Raff, W. Fleshman, R. Zak, H. S. Anderson, B. Filar, and M. McLean, "Classifying sequences of extreme length with constant memory applied to malware detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 11, 2021, pp. 9386–9394.

[5] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[6] X. Ling, L. Wu, J. Zhang, Z. Qu, W. Deng, X. Chen, C. Wu, S. Ji, T. Luo, J. Wu *et al.*, "Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art," *arXiv preprint arXiv:2112.12310*, 2021.

[7] D. Maiorca, B. Biggio, and G. Giacinto, "Towards adversarial malware detection: Lessons learned from pdf-based attacks," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–36, 2019.

[8] D. Park and B. Yener, "A survey on practical adversarial examples for malware classifiers," in *Reversing and Offensive-oriented Trends Symposium*, 2020, pp. 23–35.

[9] D. Li, Q. Li, Y. Ye, and S. Xu, "Arms race in adversarial malware detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–35, 2021.

[10] ——, "A framework for enhancing deep neural networks against adversarial malware," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 1, pp. 736–750, 2021.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018. [Online]. Available: https://arxiv.org/abs/1802.05365

[13] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[15] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[16] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[17] B. Roziere, M.-A. Lachaux, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 601–20 611, 2020.

[18] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 363–376.

[19] Z. L. Chua, S. Shen, P. Saxena, and Z. Liang, "Neural nets can learn function type signatures from binaries," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 99–116.

[20] W. Guo, D. Mu, X. Xing, M. Du, and D. Song, "{DEEPVSA}: Facilitating value-set analysis with deep learning for postmortem program analysis," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1787–1804.

[21] X. Li, Y. Qu, and H. Yin, "Palmtree: learning an assembly language model for instruction embedding," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3236–3251.

[22] F. Artuso, M. Mormando, G. A. Di Luna, and L. Querzoni, "Binbert: Binary code understanding with a fine-tunable and execution-aware transformer," *arXiv preprint arXiv:2208.06692*, 2022.

[23] P. Liguori, E. Al-Hossami, D. Cotroneo, R. Natella, B. Cukic, and S. Shaikh, "Can we generate shellcodes via natural language? an empirical study," *Automated Software Engineering*, vol. 29, no. 1, pp. 1–34, 2022.

[24] S. Rothe, S. Narayan, and A. Severyn, "Leveraging pre-trained checkpoints for sequence generation tasks," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 264–280, 2020.