Attacks on machine learning models based on the PyTorch framework

T. M. Bidzhiev[1,*] and D. E. Namiot[1,†]

[1]*Lomonosov Moscow State University*

In recent years, neural networks have gained recognition for their potential in solving information technology problems. However, their training process demands significant computational resources, leading to the emergence of cloud-based services as a solution. Nevertheless, these services also introduce new cybersecurity risks. This research investigates a novel attack method that exploits neural network weights to distribute hidden malware. By strategically embedding malware within the network's weight parameters, it becomes possible to discreetly transmit malicious code, evading detection by traditional antivirus and steganography techniques. The study explores seven distinct methods for embedding and activating malware in neural networks and examines four trigger types for initiating malware activation. Additionally, the paper presents a new open-source framework that offers pre-built implementations of these methods. This framework facilitates automated code injection into the weight parameters of neural network models. Rigorous testing has confirmed its effectiveness and reliability, enabling researchers to delve into this emerging attack vector and develop countermeasures.

Keywords: Neural networks, Malware, Steganography, Triggers

I. INTRODUCTION

The remarkable progress and widespread implementation of machine learning in diverse domains have raised important questions regarding the security and reliability of machine learning models[1, 2].

*Electronic address: temirlanbid@gmail.com

†Electronic address: dnamiot@gmail.com

One of the main security issues in machine learning is the vulnerability of models to all kinds of attacks[3, 4]. A new trend in modern machine learning systems is the increasing number of attacks aimed at introducing malware into neural networks[5–7]. This form of attack poses a serious threat to the security and reliability of models, as attackers can use them to perform malicious actions, bypassing traditional security mechanisms. Such attacks can lead to covert activation of malicious features, leakage of sensitive information, or misclassification of data, undermining accuracy in machine learning models[8]. Therefore, understanding, detecting, and defending against attacks of introduction malware into neural networks are becoming important cybersecurity challenges.

Malware infiltration techniques are used when supplying off-the-shelf models to the end user via MLaaS (Machine Learning as a Service) services[9]. Often consumers who are not machine learning experts work with MLaaS series without an understanding of learning, testing, and data processing. As a rule, the most important criterion for such users is the accuracy of the model. An attacker can take advantage of this and inject malware into the deep neural network model without the user's suspicion.

An overview of methods for injecting malware directly into machine learning models is given in our paper [10].

II. PROBLEM STATEMENT

The goal of this paper is to study and analyze machine learning attacks focused on introducing malicious code into neural networks by developing a specialized framework[11] that automates the process of introducing malicious code into neural network weights[12].

This framework allows researchers and specialists to experiment and test the robustness of their machine learning models and develop and apply countermeasures to defend against such attacks. The framework provides flexibility and scalability, allowing for customization and adaptation of malware injection methods depending on specific requirements and usage scenario.

By conducting an in-depth analysis of existing methodologies and exploring software implementation approaches, this paper aims to contribute to the field of machine learning security and provide practical insights into protecting machine learning models from code injection attacks on model weights.

In the following sections, we will delve into the theoretical background, research methodology, and experimental evaluation of the proposed framework with the ultimate goal of improving the security and reliability of machine learning systems.

III. METHODOLOGY

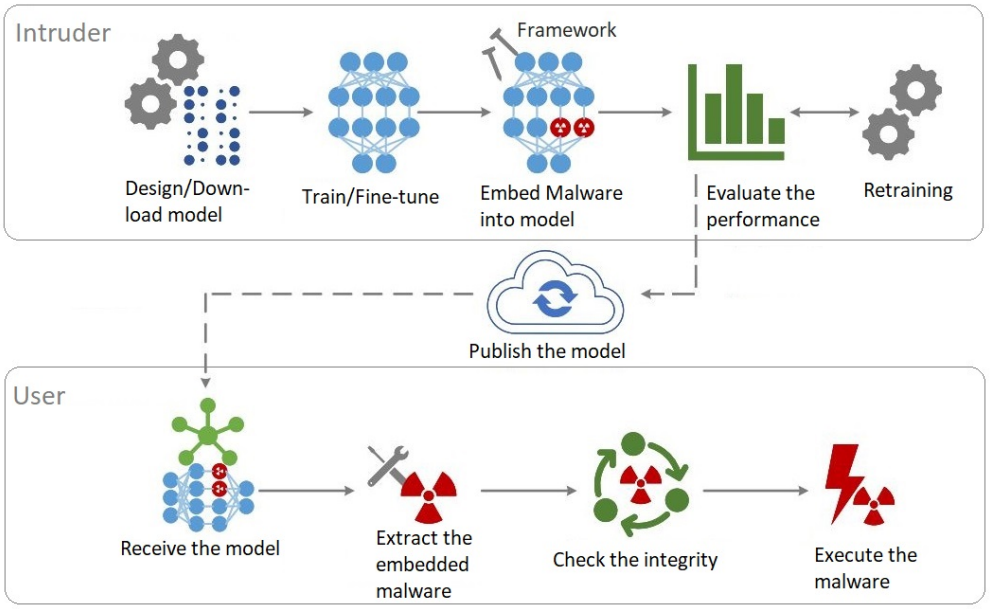

FIG. 1: General user and attacker interaction scenario.

The figure 1 illustrates a typical scenario involving the interaction between a user and an attacker. In this scenario, a user, who intends to utilize a neural network model for business purposes, initiates the process by selecting the desired model architecture. Subsequently, the user proceeds with training the model using MLaaS providers or obtains a pre-trained model from various sources.

In the specific scenario where the attacker assumes the role of an MLaaS service, it is assumed that the attacker does not possess the capability to modify the architecture of the resulting neural network. However, the attacker retains the ability to manipulate the weight parameters of the network. This allows the attacker to introduce malicious modifications to the model without altering its fundamental structure.

In order to implement all the goals, the attacker needs to perform the steps according to the figure 1. In this process, the attacker begins by acquiring a neural network model,

which can either be provided by the user or designed by the attacker themselves. They then proceed to train the model to the desired level of accuracy, a step that can be executed by the attacker or outsourced to MLaaS providers. Following successful training, the attacker prepares and introduces the malware into the model while monitoring and controlling the loss of model accuracy, using methods detailed in this paper. To deploy and activate the malware, the attacker creates a trigger, employing techniques discussed within the same research. Once the model is fully prepared, the attacker can employ additional techniques, such as "chain pollution"[13], to disseminate it to public repositories or other locations.

Next, the user gets his neural network with embedded malware, which will be extracted and run, when the trigger is activated.

IV. METHODS OF INTRODUCING MALWARE

A. Introducing malicious bytes into neurons

According to the IEEE[14] standard, a floating-point number is 32 bits wide, where the first bit is allocated for the sign of the number, the next 8 bits are allocated for the exponent, and the last 23 bits are the mantissa. The result is the number $\pm 1.m \times 2^n$ in binary form, where m is the mantissa of the number, n is the exponent. Such a number belongs to the range from 2^{-127} to $2^{127} - 1$. Its exponent is responsible for its magnitude, i.e. by keeping the first few bytes of the number, the remaining part can be replaced by malware bytes, keeping a small difference with the original number.

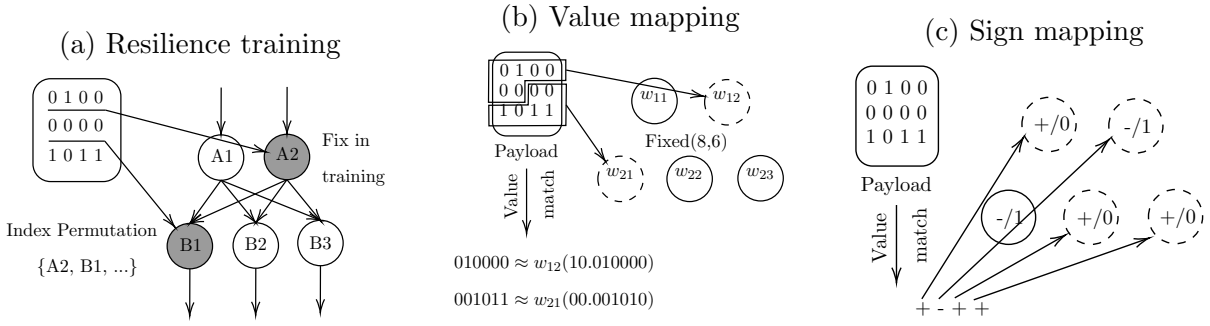

FIG. 2: Methods of introducing malware [5]

B. StegoNet methods

The article StegoNet[5] suggests four methods of introducing malware. Let's review them.

a. LSB substitution

This approach is based on the use of LSB (Least Significant Bits) replacement[7] as employed in steganography techniques[15]. It involves selecting the number of neurons and parameters that will accommodate the malware. The malware's total bit-length is divided into segments, each aligned with the chosen LSB substitution length, and these segments are then written to the initial parameters of the model. Subsequently, the remaining model parameters are used to store the complete malware code.

This solution is not applicable to highly compressed neural network models. For example, the MobileNet[16] size is only 4 MB with 4 million 8-bit parameters. These compressed models quickly lose accuracy even with minor parameter changes. This method is not suitable for compressed models.

b. Resilience training

Removal of some set of neurons from the topology of neural network model can lead to a significant decrease in the accuracy, but the parameters connecting the remaining neurons can be adjusted (retrained) to achieve the original accuracy. Based on this intuition, the Resilience training technique was proposed.

As shown in figure 2a, the method implies a direct replacement of all bits of the selected parameters with malware segments. Such neurons (i.e. with modified parameters) will not be updated during retraining. It is expected that after training, the accuracy of the model will be restored, which will allow to hide the presence of injected software.

c. Value mapping

Suppose we have a model with 8-bit fixed-point numbers with 6 bits after the decimal point. To facilitate value comparison, the binary code of the malware is initially divided into segments of a length corresponding to the number of bits after decimal point. Then, for each segment, a full exhaustive model parameter search is performed to match (or replace) the same (or close) value of the fractional parameter bits, as an example, shown in 2b.Finally, we map the code segment to the corresponding parameter, replacing the fractional bits of the parameter with the code segment if necessary.

d. Sign mapping

The sign-matching method uses a similar "full search and match" rule based on the sign bit of the model parameters. As shown in 2c, sign matching will go through the model parameters and map a parameter sign bit to each individual bit in the malicious code, for example 0 is mapped to a + parameter sign, thus ultimately mapping the code to a sequence of sign bits for the corresponding parameters. It is necessary to keep a vector of mapped parameters.

C. EvilModel methods

The article EvilModel[6] proposes three other methods. Let's review them.

a. MSB reservation

Since the most important exponential part of the parameter is mostly in the first byte, the first byte is the most significant byte for determining the value of the parameter. Therefore, it is possible to leave it unchanged and inject the malware into the next three bytes. Thus, the parameter values are still within reasonable limits.

b. Fast substitution

If we replace the parameters with three bytes of malware and the first byte of prefix 0x3C or 0xBC based on the parameter sign, most parameter values would still be within reasonable limits. Compared to the MSB method, this method may have a greater impact on model performance, but since it does not need to parse parameters in the neuron, it will run faster.

c. Half substitution

Similar to the MSB reservation, if you keep the first two bytes unchanged, instead of one, and change the other two bytes, the value of that number will fluctuate in a smaller range. However, since only two bytes are replaced in the parameter, this method can introduce less payload than the previous two methods.

V. COMPARISON OF EMBEDDING METHODS

Let's compare the accuracy of different models before and after the implementation of all the methods above. See [5, 6] and the table I.

As you can see from this table, the naive LSB replacement method can maintain good test-

ing accuracy on medium neural networks; this fact does not apply to small neural networks. For example, it leads to a significant decrease in accuracy (i.e., a sharp drop to $\approx 0.1\%$) in high compression neural network models due to limited data accuracy and reduced number of parameters.

In contrast, Resilience training can relatively better support malware input on small neural networks. For small malware such as EquationDrug, ZeusVM, and Cerber[17], it can maintain testing accuracy at the same level as the original, even in the smallest MobileNet[16] (4.2 MB) and SqueezeNet[18] (4.6 MB). However, MobileNet's accuracy dropped significantly from 66.7% to 0.7% as the malware size increased from 0.59MB (Cerber) to 3.35MB (WannaCry). It can be observed that the upper bound of the malware-to-model size ratio for the Resilience training method is $\approx 15\%$ without a decrease in accuracy.

However, such a problem has been eliminated with methods based on "full search and mapping". For highly compressed neural network models such as "Comp.Alexnet"[19], the model parameters are extremely compressed, the method may be less efficient for such models, i.e. the parameters are also compressed. A similar tendency can be found in the Sign mapping methodIn general, however, Sign-mapping can always maintain the original testing accuracy for all applicable cases.

For the MSB reservation method, due to the redundancy of neural network models, when malware is embedded, testing accuracy is unaffected for models of large size (>200MB). Accuracy increased slightly in some cases (e.g., Vgg 16[20] with NSIS), as also noted in Stegonet [5]. When malware is implemented using MSBs, accuracy decreases as the size of the embedded malware increases for medium- and small-sized models. For example, accuracy decreases by 5% for medium-sized models such as Resnet50 with Mamba. Theoretically, the maximum order of embedding (i.e., the ratio of malicious code size to model size) of the MSB reservation method is 75%. In the experiment, the upper bound of the embedding order without a significant decrease in accuracy is 25.73% (Googlenet with Artemis).

The performance of the Fast substitution method is similar to the MSB reservation method, but unstable for small models. When larger malware is introduced into a medium- or small-sized model, the accuracy of the model decreases significantly. For example, for Mobilenet with VikingHorde, testing accuracy drops dramatically to 0.108%. This shows that fast substitution can be used as a substitute for the MSB reservation method when the model is large or the task is time-consuming. In the experiment, the embedding order

turned out, without a significant decrease in accuracy, to be 15.9% (Resnet18 with Viking Horde).

The Half substitution method outperforms all other methods. Due to the redundancy of neural network models, the accuracy after embedding all sizes of code practically does not degrade, even when almost half of the model has been replaced with malicious bytes, the accuracy varies within 0.01% of the original. A small Squeezenet size (4.74MB) can embed a 2.3MB Mamba virus sample with accuracy increasing by 0.048%. Theoretically, the maximum order of embedding is 50%. In the experiment a close to theoretical value of 48.52% was achieved (Squeezenet with Mamba).

VI. METHODS OF MALWARE ACTIVATION

A. Triggers

The StegoNet article suggests three different triggers: Logits trigger, Rank trigger, Fine-tuned Rank Trigger.

The Logits Trigger method involves memorizing the logits outputs for certain input data - triggers. Then, when one of the inputs we have chosen as a trigger is fed, the outputs of the logits will match, and the malicious code will be extracted and activated. In theory, this is not possible, because there is a very small chance of feeding exactly the same input sample, and the logit outputs (floating-point numbers) must match completely.

Therefore, the Rank trigger method would be more useful. The difference is to compare logit outputs instead of their ranks, i.e. indexes in a sorted array of logits. For example, let the last layer have dimension 3 and the logit output is $\{p_1, p_2, p_3\} = \{0.5, 0.2, 0.4\}$. But because of the variety of inputs, we got the output $\{0.55, 0.13, 0.42\}$. The logit ranks will be $r = \{p_1, p_3, p_2\}$, and they will coincide.

The Fine-tuned Rank Trigger method involves selecting a triggered input sample, augmenting it with various variations, and retraining the model on these augmented samples. However, instead of using the original loss function that depends on logit values, the loss function based on logit ranks is employed. To do this, we will have to manually set the target value of logit ranks. Let x be the augmented input data, h^r be the logit rank label

set to it, if no logit is considered its value is set to 0.

$$arg\min_{w} \frac{1}{n} \sum_{1}^{n} \mathcal{L}(f_w(x), h^r).$$

After the trigger is activated, the malware is assembled into a single code. Its hash sum is then compared with the pre-stored hash sum of the unsegmented malware. If they match, the malware is launched.

B. Activation

If the neural network model is accompanied by third-party software, such as a model exploitation program, then all the necessary code for checking triggers and activating malware on the target device can be injected into it. This is the simplest case, let's look at the others.

We first assumed that the model is trained on an untrusted source, i.e., an attacker, and he has all the model data, a white box attack. The model is passed through the network in serialized form[21], then deserialized. With an attack like "insecure deserialization"[22, 23], the attacker can modify the activation functions in the model. For example, instead of softmax use a modified version with trigger checking and malware deployment.

Another approach is to exploit vulnerabilities in libraries and executable environments. For example CVE-2018-6269, CVE-2017-12852.

VII. FRAMEWORK

A. Review

The goal of our framework[11] is to develop an off-the-shelf solution to the malware embedding into neural networks, allowing malicious code to be integrated seamlessly into the network architecture. Using the unique characteristics of neural networks and their widespread use in various applications, our framework aims to explore the potential risks and vulnerabilities associated with these models.

Our framework provides a comprehensive solution that allows researchers and security professionals to study the behavior of neural networks when exposed to embedded malware. This opens the door to analyzing the impact of malicious attacks on network performance, identifying vulnerabilities and developing robust defense mechanisms.

Key Features:

1. **Introduction of malware**: Our framework incorporates various techniques for introducing malware into the neural network structure by changing its weighting factors, providing seamless integration without compromising the overall functionality of the network.

2. **Evaluation opportunities**: Our platform provides tools to assess the impact of embedded malware on network performance, i.e., accuracy degradation metrics.

3. **Flexibility and compatibility**: The framework is designed to be compatible with the PyTorch[24] deep learning framework, allowing easy integration into existing neural network architectures without modification.

Overall, our platform enables cybersecurity researchers and practitioners to better understand the consequences of malware introduction into neural networks. By providing a robust and flexible platform, it facilitates the search for effective countermeasures and the development of more resilient and secure neural network models.

B. Experimental evaluation

To test the fall in accuracy of the models before and after malware injection using our framework, a series of experiments with different neural network architectures[25] were conducted.

1. **Dataset:** For testing the model and framework, the ImageNet dataset [26] was utilized. This dataset comprises 1.2 million 3-channel images with dimensions of 224×224 pixels, representing 1000 distinct object categories.

2. **Methodology:**

   - We performed experiments using several pre-trained neural network models.
   - For each experiment, different sizes of malware were injected using our framework. The malware examples were taken from the repositories[17, 27].
   - The performance of the modified models was compared with their original models and the effect of introduced malware on model accuracy was assessed.

TABLE I: The accuracy of models after malware injection into their weights using the framework. Cases with a significant drop in model accuracy are highlighted in bold.

Method	**Model**	**Base**	**EquationDrug** 372KB	**ZeusVM** 405KB	**NSIS** 1.7MB	**Mamba** 2.30MB	**WannaCry** 3.4MB	**VikingHorde** 7.1MB	**Artemis** 12.8MB
LSB Substitution	Alexnet	52.8%	52.8%	52.8%	52.8%	52.8%	52.8%	52.8%	52.8%
	Resnet101	76.7%	76.7%	76.7%	76.4%	76.3%	76.2%	75.7%	74.9%
	Inception	68.0%	67.9%	68.0%	68.1%	68.0%	67.9%	67.8%	67.1%
	Resnet50	76.0%	76.0%	76.1%	76.0%	76.3%	75.9%	76.2%	75.2%
	Googlenet	67.1%	66.8%	66.9%	66.7%	65.9%	65.7%	-	-
	Resnet18	67.8%	67.9%	67.8%	67.3%	68.0%	67.5%	**58.4%**	-
	Mobilenet	70.9%	**0.1%**	**0.1%**	**0.1%**	**0.1%**	-	-	-
	Squeezenet	54.9%	**0.1%**	**0.1%**	-	-	-	-	-
MSB Reservation	Inception	68.0%	68.0%	68.2%	67.7%	67.0%	68.1%	61.1%	62.7%
	Resnet18	67.8%	67.7%	67.4%	67.3%	67.0%	66.3%	66.2%	60.9%
	Mobilenet	70.9%	71.1%	69.2%	68.1%	67.0%	63.8%	**0.7%**	-
Fast Substitution	Inception	68.0%	68.0%	67.7%	68.0%	68.1%	67.2%	67.9%	68.0%
	Resnet18	67.8%	67.7%	67.3%	67.2%	67.0%	67.6%	66.2%	61.2%
	Mobilenet	70.9%	70.8%	70.9%	65.7%	**59.8%**	**40.7%**	**1.6%**	-
Half Substitution	Inception	68.0%	68.0%	68.0%	68.0%	68.0%	68.0%	68.0%	68.0%
	Resnet18	67.8%	67.8%	67.8%	67.8%	67.8%	67.8%	67.8%	67.8%
	Mobilenet	70.9%	70.9%	69.9%	69.3%	66.0%	67.7%	**52.0%**	-
Resilience Training	Inception	68.0%	68.4%	67.6%	67.8%	67.3%	68.3%	67.7%	67.8%
	Resnet18	67.8%	67.5%	67.7%	68.0%	67.9%	67.2%	67.3%	68.0%
	Mobilenet	70.9%	**54.9%**	**20.4%**	**0.4%**	**0.4%**	**0.7%**	-	-
Value-Mapping	Inception	68.0%	68.0%	68.0%	67.4%	67.7%	67.5%	67.2%	66.2%
	Resnet18	67.8%	67.4%	67.4%	67.2%	67.4%	67.9%	67.4%	67.2%
	Mobilenet	70.9%	70.1%	70.7%	**60.1%**	**53.1%**	-	-	-
Sign-Mapping	Inception	68.0%	68.0%	68.0%	68.0%	-	-	-	-
	Resnet18	67.8%	67.8%	67.8%	67.8%	-	-	-	-
	Mobilenet	70.9%	-	-	-	-	-	-	-

3. **Results and analysis:**

   Table I shows the accuracy of the models as a result of malware injection using our framework.

   We chose the most frequently used models and malware samples to compare the framework's capabilities. Since large models have a large capacity, the implementation re-

sults were shown only for "LSB substitution" methods, and for the rest of the methods mostly medium and small models are compared. Note that no model retraining was used for the "Resilience Training" method.

- The results of the experiment demonstrated the effectiveness of the framework in introducing malware into neural networks.
- The modified networks successfully maintained accuracy in most classification cases on ordinary input data, exhibiting the desired evasive behavior.
- Experimental results revealed a tradeoff between the capacity of the models and the flow of classification accuracy.

VIII. COUNTERMEASURES

Protecting neural networks from the introduction of malware is critical to ensuring their integrity and reliability. Several countermeasures can be applied to reduce the risks associated with this type of attack[28]. The following countermeasures are commonly used:

1. **Changing the network architecture:** It is possible to change the model structure or its parameters so that the hash value of the deployed model does not coincide with the stored one. This can be achieved by changing the network architecture, adding random layers, or changing the weighting parameters[29].

2. **Using reliable model supply channels:** By choosing reputable and trustworthy MLaaS providers, organizations can significantly reduce the likelihood of acquiring malware-infected models[30].

3. **Regular model updates:** Keeping neural network models up to date with the latest security patches and updates is critical to protect against emerging threats. Regular model updates with improved architecture, robust learning algorithms, and enhanced security measures can help prevent and detect malware infiltration attempts.

4. **Monitoring models:** Continuous monitoring of deployed models is necessary to detect any unusual behavior or deviations from expected patterns. Techniques such as model self-analysis, anomaly detection, and run-time analysis can help identify potential malware introductions and initiate appropriate responses[31].

IX. CONCLUSION

This work introduces a novel framework for embedding malware into neural networks, leveraging the weight parameters of neurons as carriers of malicious information. By integrating various methods and components, the framework demonstrates the ability to embed malicious code into neural network models, highlighting the potential security risks associated with deploying such models.

Experimental evaluation of this framework on various neural network architectures, has demonstrated its effectiveness in successfully introducing malware while maintaining the functionality and performance of the model. The findings underscore the need for robust security measures to address the growing level of threats posed by embedded malware in machine learning systems.

We discussed the architecture and components of our framework, including the malware injection methods used, the integration process, and model testing.

In conclusion, our platform proves the vulnerabilities present in machine learning systems and emphasizes the need for proactive security measures to ensure the integrity, reliability, and resilience of these systems. By understanding and mitigating the risks associated with embedded malware, we can improve the security of machine learning applications and help create a more secure digital landscape.

---

[1] E. I. Namiot Dmitry and O. Pilipenko, *"on trusted ai platforms."*, internation Journal of Open Information Technologies 10.7 (2022): 119-127.

[2] V. Kostyumov, *"a survey and systematization of evasion attacks in computer vision."*, international Journal of Open Information Technologies 10.10 (2022): 11-20.

[3] I. Eugene, D. Namiot, , and I. Chizhov, *"attacks on machine learning systems-common problems and methods."*, internation Journal of Open Information Technologies 10.3 (2022): 17-22.

[4] D. K. Marc Ph. Stoecklin co-authored by Jiyong Jang, *Deeplocker: How ai can power a stealthy new breed of malware* (2018).

[5] T. Liu, Z. Liu, Q. Liu, W. Wen, W. Xu, and M. Li, in *Annual Computer Security Applications Conference* (Association for Computing Machinery, New York, NY, USA, 2020), ACSAC '20, p. 928–938, ISBN 9781450388580, URL https://doi.org/10.1145/3427228.3427268.

[6] Z. Wang, C. Liu, X. Cui, J. Yin, and X. Wang, *Evilmodel 2.0: Bringing neural network models into malware attacks* (2021), 2109.04344.

[7] T. Liu, W. Wen, and Y. Jin, in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2018), pp. 227–230.

[8] S. Stefnisson, *Evasive malware now a commodity* (2018), URL `https://www.securityweek.com/evasive-malware-now-commodity`.

[9] *Mlaas*, URL `https://en.wikipedia.org/wiki/Machine_learning_as_a_service`.

[10] T. Bidzhiev and D. Namoit, *"research of existing approaches to embedding malicious software in artificial neural networks."*, international Journal of Open Information Technologies 10.9 (2022), pp 21–31 (in Russian).

[11] B. Temirlan, *Nnmalwareembedder*, `https://github.com/Temish09/NNMalwareEmbedder`.

[12] K. K. P. Michel and G. Neubig, *"weight poisoning attacks on pre-trained models."*, arXiv preprint arXiv:2004.06660 (2020).

[13] T. H. N. R. Lakshmanan, *A large-scale supply chain attack distributed over 800 malicious npm packages*, URL `https://thehackernews.com/2022/03/a-threat-actor-dubbed-red-lili-has-been.html`.

[14] I. C. Society, *Ieee 754-2019 - ieee standard for floating-point arithmetic* (2019), URL `https://standards.ieee.org/ieee/754/6210/`.

[15] K. S. D. Neeta and D. Jacobs, *"implementation of lsb steganography and its evaluation for various bits,"*, in 2006 1st International Conference on Digital Information Management, 2007, pp. 173–178.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, CoRR **abs/1704.04861** (2017), 1704.04861.

[17] ytisf, *thezoo - a live malware repository*, `https://github.com/ytisf/theZoo` (2021).

[18] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, arXiv preprint arXiv:1602.07360 (2016).

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Advances in neural information processing systems **25**, 1097 (2012).

[20] K. Simonyan and A. Zisserman, arXiv preprint arXiv:1409.1556 (2014).

[21] Guido van Rossum (original author), Python Software Foundation (maintainer), *pickle – python object serialization*, Python Software Foundation, Python Documentation (2021).

[22] T. of Bits, *Fickling*, `https://github.com/trailofbits/fickling` (2021).

[23] P. Ltd, *Insecure deserialization*, URL `https://portswigger.net/web-security/deserialization`.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., *Pytorch: An imperative style, high-performance deep learning library*, `https://pytorch.org/` (2019), accessed: 2022-05-04.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, Proceedings of the IEEE conference on computer vision and pattern recognition pp. 1–9 (2015).

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, in *2009 IEEE conference on computer vision and pattern recognition* (Ieee, 2009), pp. 248–255.

[27] InQuest, *malware-samples*, `https://github.com/InQuest/malware-samples` (2021).

[28] G. Y. et al., *"strip: A defence against trojan attacks on deep neural networks."*, proceedings of the 35th Annual Computer Security Applications Conference. 2019.

[29] G. Y. et al., *"backdoor attacks and countermeasures on deep learning: A comprehensive review."*, arXiv preprint arXiv:2007.10760 (2020).

[30] P. S. Z. Wu and P. D. Christofides, *"cybersecurity in process control, operations, and supply chain."*, computers & Chemical Engineering (2023): 108169.

[31] R. C. et al., *"live trojan attacks on deep neural networks."*, proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. 2020.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, D. E. Namiot, for His invaluable guidance, support, and mentorship throughout the duration of this project. I am truly grateful for His patience, encouragement, and willingness to share knowledge.

I would also like to extend my thanks to Department of Information Security of the Faculty of Computational Mathematics and Informatics of the Moscow State University for their assistance and contributions to this project.

This research has been supported by the Interdisciplinary Scientific and Educational School of Moscow University "Brain, Cognitive Systems, Artificial Intelligence".

X. REVIEWERS REMARKS

The main remark was about the size of article. It was 37 pages, and now it is reduced to 15. All the corrections are listed after reviews and answers.

A. Review 1

2: (accept)

The paper presents a framework to "manipulate" the behavior of neural networks in order to deploy malware in the host's computer system.

strong points of the paper:

1. the introduction that explains different kinds of attacks is very complete. In general terms, bibliography seems to be complete.

2. the writing is concise and correct.

3. the structure of the paper is good and according to the literature's standards.

4. the experiments are sound.

In general terms, the paper is very well written, and the material is of interest.

potentially weak points:

1. the explanations (specially in section "COMPARISON OF EMBEDDING METHODS") may be a little bit hard to explain. In general, some technical explanations without examples may be hard for people not used to cybersecurity. Obviously, this is a problem that has little remedy.

2. Some punctuation marks are not correct. There is a lot of expressions quoted between ¡¡ ¿¿.

3. I do not know if having figures and tables at the end of the paper is a good idea. Having them next to the text where they appear would probably help the reader.

4. although the subject is connected to AI, it could be argued that it does not fit completely into the aims of this conference. But my point of view is that this material is of interest to the community.

These "weak" points are not as strong as the "strong" points, therefore, I recommend to accept this paper.

1. Answer

Thank you for all the kind words! Regarding to weak points:

1. The "COMPARISON OF EMBEDDING METHODS" section has been adjusted, but I'm unable to assess whether it has improved or declined in quality.

2. All the punctuation marks were fixed to this "quotes".

3. Figures and tables have become scattered throughout the article.

4. I hope so, thank you!

B. Review 2

2: (accept) The submission is more, than a conference paper. First, its 37 pages is much over the regular 15p full paper size. Looking at the paper structure one can see that first 14 pages are a kind of a review. Next, we have an almost same size of proposal, followed by another 10 pages of appendix.

The idea of a paper is good, and it is given in a good manner, but I'm sure the authors should split this submission in two, or submit it in a full size to a proper journal as an extended version.

To fit in the conference limits, I would suggest to slash the introdution and appendix.

From a technical point of view, the work is promising.

1. Answer

Thank you very much! The articles size is reduced to 15 pages. All the corrections are listed in the end.

C. Review 3

-1: (weak reject) In this paper authors present a novel framework for investigating security leaks in deep neural networks. It allows one to study various methods of malware injection and triggering, which are described in the paper. Overview of recent results in this area is provided. Since malware program needs space, it's injection into neural network weights may lead to performance degradation of the model. Authors provide data on accuracy decrease

of several models corresponding to injection of different malware files. Results show that there is clear dependency between complexity of program to be hidden, size of a network, it's compression characteristics and the degradation in performance. Experiments conducted by authors are methodologically correct, and results obtained seem to be quite meaningful. The framework presented may prove to be useful for cybersecurity researchers and ML engineers.

Though quite interesting, currently the paper does not meet DAMDID formatting rules, is 37 pages long and contains appendix. In order to be accepted it should be completely reformatted in strict conformance with the LNCS Word or Latex format.

1. Answer

Thank you very much! The size of the article was reduced to 15 pages and no longer contains appendix. As far as I understand, the article meets the conference requirements now.

XI. ARTICLE CORRECTIONS

A. Abstract

1. The sentences 6, 7 "The study explores ... initiating malware activation" were rewritten into one sentence.

2. Added keywords.

B. Introduction

paragraph 2. Last sentence deleted.

paragraph 3. deleted.

paragraph 4. First sentence was left, the rest deleted.

paragraph 4-5. merged.

The last sentence was added.

C. Problem statement

paragraph 3. deleted.

paragraph 4. deleted.

D. Methodology

Added figure.

paragraph 1. MLaaS full name deleted.

paragraph 1. Last sentence deleted. "For example ..."

paragraph 3. Enumeration is rewritten into paragraph.

subsection A. deleted.

E. Methods of introducing malware

paragraph 1. Deleted.

subsection A. The reference for deleted figure was deleted.

subsection A. Added figure.

subsection B. Line 1: removed bullets list. Added "Let's review them." instead.

subsection B. paragraph A. paragraph 1. Deleted. Removed last two sentences and pseudocode 1.

subsection B. paragraph A. paragraph 2. The beginning of the paragraph is deleted till "However, this solution ..." which is replaced by "This solution ...". The next sentence is deleted "As shown in table ...".

subsection B. paragraph B. Enumeration list was deleted.

subsection B. paragraph B. the half of penultimate paragraph and the last paragraph were deleted.

subsection B. paragraph C. First sentence was deleted.

subsection B. paragraph C. paragraphs 2-3. The last sentences were deleted.

subsection B. paragraph C. paragraph 4. The last two sentences were deleted.

subsection C. Line 1: removed bullets list. Added "Let's review them." instead.

subsection C. paragraph A. the last several sentences were deleted starting from "For example ..."

subsection C. paragraph B. The first several sentences were deleted till "Therefore, ...". The word was deleted and new sentence started by "If we replace ...". And the last sentence was deleted.

subsection C. paragraph C. paragraph 1. Last sentence was deleted. "For example ...".

subsection C. paragraph C. The last sentence was deleted.

F. Comparison of embedding methods

paragraph 2. Deleted. All the references to deleted tables were deleted. Instead there is a reference to corresponding articles.

G. Methods of malware activation

subsection Triggers. First two paragraphs were deleted.

subsection Triggers. Itemize list was deleted.

subsection Triggers. paragraph 6. Deleted.

subsection Triggers. paragraph 7. The last sentence was deleted "For example, ..."

subsection Activation. Reference for CVE table was deleted.

subsection Activation. CVE explanations were deleted. Instead their numbers are just given.

H. Framework

subsection Review. In enumeration points 2 and 5 are deleted.

subsection Architecture and components. Deleted.

subsection Experimental evaluation. paragraph 1. The last sentence is deleted.

subsection Experimental evaluation. Table was moved from appendix.

subsection Experimental evaluation. Enumeration list. Point Methodology. The sentences containing references for deleted tables were deleted.

subsection Experimental evaluation. Enumeration list. Point Results and analysis. The explanation for the table "The dash mark ..." was deleted.

subsection Experimental evaluation. Enumeration list. Point Discussion and Implications. Deleted.

subsection Limitations and future work. Deleted.

I. Countermeasures

Enumeration list. Point 1. Last several sentences were deleted.

Enumeration list. Point 2. The first sentence was left, the rest - deleted.

Enumeration list. Point 3. Deleted.

Enumeration list. Point 6. Deleted.

The last two paragraphs were deleted.

J. Conclusion

Paragraph 2. "including AlexNet, ResNet, and MobileNet, among others" was deleted.

Paragraph 4. Deleted.

K. Bibliography

The bibliography size was reduced to 30 entries.

L. Appendix

FIG 1. Size reduced to 90%.

FIG 2. Deleted.

FIG 3. Deleted.

TABLE 1. Deleted.

Algorithm 1. Deleted.

TABLE VI. Deleted.

TABLE VII. Deleted.

TABLE VIII. Deleted.

TABLE IX. Deleted.

TABLE X. Deleted.

TABLE XI. Deleted.

TABLE XII. Moved to section "Framework".

Appendix. Deleted.