Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 699–704
DOI: 10.15439/2025F9981
ISSN 2300-5963 ACSIS, Vol. 43

AI-MTD: Zero-Trust Artificial Intelligence Model Security Based on Moving Target Defense

Daniel Gilkarov
0009-0008-9274-802X
Department of Computer and Software Engineering
Ariel Cyber Innovation Center
Ariel University, Israel
daniel.gilkarov1@msmail.ariel.ac.il

Ran Dubin
0000-0002-2055-2211
Department of Computer and Software Engineering
Ariel Cyber Innovation Center
Ariel University, Israel
rand@ariel.ac.il

***Abstract*—This paper examines the challenges in distributing AI models through file transfer mechanisms. Despite advancements in security measures, vulnerabilities persist, necessitating a multi-layered approach to mitigate risks effectively. The physical security of model files is critical, requiring stringent access controls and attack prevention solutions. This paper proposes a novel solution architecture that protects the model architecture and weights from attacks by using Moving Target Defense (MTD), which obfuscates the model, preventing unauthorized access, and enabling detection of changes to the model. Our method is shown to be effective at detecting alterations to the model, such as steganography; it is faster than encryption (0.1 seconds to obfuscate vs. 18 seconds to encrypt for a 2500 MB model), and it preserves the accessibility of the original model file format, unlike encryption. Finally, our code is available at https://github.com/ArielCyber/AI-model-MTD.git.**

I. Introduction

THE swift evolution of Artificial Intelligence (AI) technology has made it a top priority for cybercriminals looking to obtain confidential information and intellectual property or cause damage in different ways. These malicious individuals may try to exploit AI systems for their own gain, using specialized tactics alongside conventional IT methods. In particular, attackers may wish to gain access to a trained AI model for any number of reasons: to exfiltrate valuable data from victims, including stealing trained AI model data, to tamper with the model, for example, implanting backdoors [1], etc. Given the broad spectrum of potential attack strategies, safeguards must be extensive.

While adversarial AI model security [2], [3], privacy [4] and operational security aspects of AI receive much attention [5], [6], it's equally important to address the physical file security aspects of AI models. In this day and age, an AI model is a valuable asset; training AI models is a long and extremely expensive process, hence, the resulting trained models are the sole artifacts that carry the effort put into the training process.

Figure 1 shows the simplified AI model architecture. Inside the model, we have the architecture metadata (in some cases), model weights (numeric parameters such as weights/biases), and optional metadata such as labels and notes. The model weights essentially carry the frozen state of the model after training, they represent the model's "knowledge" of the task it was trained on. Therefore, deploying trained AI models necessitates protecting the model file, and in particular, the trained weights. The model might be exposed to danger in different scenarios: in training, transit, deserialization, and inference.

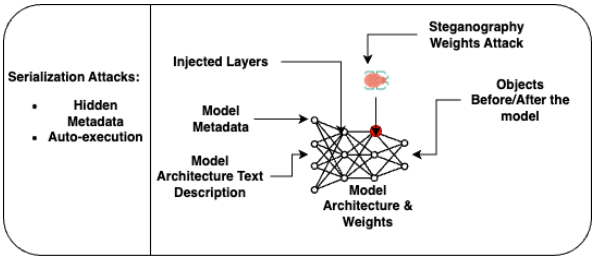

Fig. 1. Simplified AI model file architecture, and serialization attacks and steganography attacks that utilize the AI model file.

The National Security Agency (NSA) recently released a Cybersecurity Information Sheet (CSI) [5]. The CSI is intended to support National Security System owners and Defense Industrial Base companies deploying and operating AI systems designed and developed by an external entity. The CSI illustrates the steps to protect the deployment and two main points directly related to this work. The first is to validate and test the AI model. The second is protecting AI model weights. The NSA CSI suggests the following protection steps:

- Harden interfaces for accessing model weights to increase the effort it would take for an adversary to exfiltrate the weights.
  - Implement hardware protections for model weight storage, disable unnecessary hardware communication capabilities, and protect against emanation or side-channel techniques.
  - Aggressively isolate weight storage. For example, store model weights in a protected storage vault or enclave

The Open Web Application Security Project (OWASP) has been at the forefront of web application security for years. Recognizing the burgeoning significance of machine learning, OWASP introduced the "Top 10 Machine Learning Risks"

[7], a compilation that categorizes and outlines the most prevalent threats targeting AI systems, specifically about the content of the model and the threats around it. Examples of the discussed attacks are inference attacks and poisoning attacks. Of all the discussed threats, the AI Supply Chain Attacks are the most relevant to this work, while the other attacks are related to the content and integrity of the model. In the AI supply chain attack use case, an attacker modifies or replaces a machine learning library or model used by a system. This can also include the data associated with the machine learning models. MITRE Adversarial Threat Landscape for Artificial-Intelligence Systems (ATLAS™) [8] is a comprehensive knowledge base. It's designed to document adversary tactics, techniques, and case studies pertinent to machine learning (ML) systems. These details are amassed from real-world observations, demonstrations by ML red teams and security groups, and academic research. In terms of this research, ATLAS helps to understand the attack surface, but it is not a cyber solution. This work is positioned in MITRE ATLAS as a solution against user execution of unsafe ML Artifacts (AML.T0011.000) [9].

While growing, the literature on physical AI model security is still in its early stages [10], [11], [12], [13]. As the adoption of AI and Machine Learning (ML) technologies continues to rise, the need for robust security measures becomes increasingly critical. This work focuses on providing a secure solution to train, transfer, and ingest AI models.

In this work, we propose model weight separation based on Moving Target Defense (MTD) of model weights. Our MTD also provides verification and authentication mechanisms to guarantee the model's validity. By adopting an agreed-upon framework of obfuscation of serialized models and verifying models upon deserialization, model receivers can ensure they ingest models safely, whether as part of a supply chain or directly using the model.

The paper is organized as follows: Section II summarizes the paper's contribution. Section III reviews the related work. Section IV presents the MTD solution architecture. Section V describes our threat model. Section VI offers a detailed description of our dataset. In Section VII, we present our research evaluation and validation methodology. Subsequently, in Section VIII, we discuss our limitations. Finally, Section IX provides our conclusions.

II. Contribution

This work reviews the state-of-the-art physical Artificial Intelligence (AI) model security solutions. For the first time, as far as we know, we suggest a novel zero-trust solution based on MTD for AI model weights.

- We suggest a novel zero-trust MTD solution designed to bolster the robustness and security of AI models against potential threats. This approach ensures protection throughout the entire model life cycle, from its initial training phase to its final distribution. Consequently, it provides a strong defense against physical file-based attacks on the model weights and prevents unauthorized use of the trained model by obfuscating the model weights.
- We fully publish our code [14].

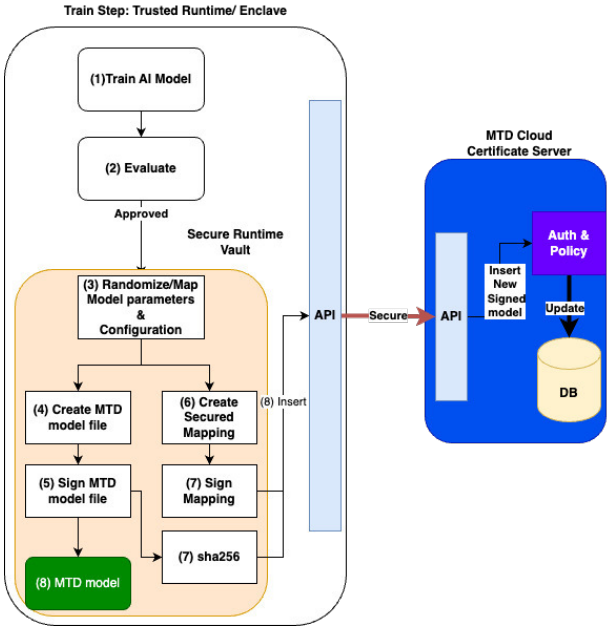

Fig. 2. MTD model creation inside a secure enclave

III. Related Work

Moving Target Defense (MTD) [15] is a proactive cybersecurity strategy that increases the complexity and cost for attackers by continuously changing the attack surface. Unlike traditional defense mechanisms, MTD introduces variability and unpredictability into the system, making it more challenging for attackers to exploit vulnerabilities. MTD encompasses techniques like IP address hopping [16], port rotation [16], virtual machine migration [17], code diversification [18], software obfuscation [19], memory randomization [20], and data randomization [21]. It is helpful in network and software security and represents a paradigm shift in cybersecurity by emphasizing dynamic and adaptive defense mechanisms. Our AI model MTD uses model weight randomization and reconstruction to prevent all model weight attacks and report an attempt of an attack.

IV. Architecture & Solutions

In this section, we describe our MTD methodology, which provides comprehensive protection against physical attacks on the model weights and prevents unauthorized use of the trained model. It is used to save and load models.

The architecture is divided into two parts: MTD creation (Fig. 2) and MTD loading (Fig. 3). If the model is not MTD protected, a fallback step optionally uses Content Disarm and Reconstruction (CDR) methods that were discussed in previous works [22] (Fig. 3).

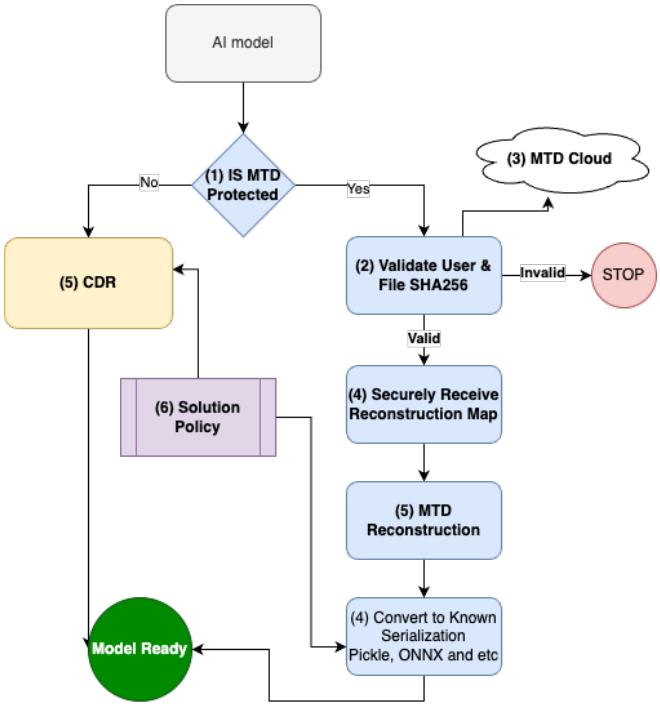

Fig. 3. MTD model loading and fallback to CDR

The model training process powered by MTD is demonstrated in Fig. 2 and occurs within a secure runtime environment or enclave [23]. Once the training is complete (1) and model evaluation passes (2), the MTD process (3) begins. The NSA [5] recommends separating the model and its weights. Therefore, the MTD algorithm randomizes the model weights, separates the model mapping from the randomized weights, and prevents unauthorized use of the trained model. To provide additional security, the MTD model's weights can be encrypted. This produces a file (4) that is protected by MTD. We sign and send this file to the cloud (5, 8) along with a mapping (6). We also sign and store the mapping in the cloud (6, 7). To further protect the model, upon saving the MTD model, we save a sha256 hash of the model weights and verify the hash on load.

Figure 3 shows how the MTD-protected model is loaded. First, it checks (1) if the model is MTD protected. If it's not, CDR (5) is used instead. If the model is protected, it's validated (2) with the MTD cloud API (3). If any modifications are detected, an alert is sent, and the process is stopped. If the model is valid, the mapping is received, and the model is reconstructed. Once the model is reconstructed, it can be saved to a file or kept in memory. It's important to note that this process assumes that if the user is compromised, it's already too late, and the focus is on securing the model and not the device. AI inference can also run inside a secure enclave. Our future work will focus on securing AI PC, which is expected to be the future of running large language models by utilizing a Neural Processing Unit (NPU) (see [24]).

V. Threat Model

To ground our security analysis, we first formalise the adversary we seek to resist, the assumptions we place on system components, and the boundaries of our evaluation. The threat model presented here guides the design choices in Section IV and underpins the evaluation in Section VII.

A. System Participants and Assets

We consider three roles:

- **Model Owner** - trains the network inside a trusted hardware enclave, applies the proposed Moving-Target Defense (MTD) to obfuscate the weights, and signs the resulting artefacts.
- **Distribution Service** - a cloud repository that stores the MTD-protected weight file and the corresponding reconstruction mapping, exposes a mutually-authenticated API, and returns the mapping only to authorised consumers.
- **Model Consumer** - fetches the protected file, verifies the signature, retrieves the mapping via the API, reconstructs the model, and performs inference.

The primary assets are: (i) the trained weights, (ii) the architecture/mapping that re-orders the weights, and (iii) the Model Owner's signing key.

B. Adversary Capabilities

The adversary ("*A*") is *active* and may:

1) Observe, intercept, replay, or tamper with any file transferred between participants (supply-chain attack, man-in-the-middle, malicious mirror).
2) Obtain *direct* read–write access to any storage location that holds the protected file at rest (e.g., compromise of a consumer endpoint or removable media).
3) Craft and deliver malicious model artefacts that embed steganography, hidden payloads, or altered weights in an attempt to bypass verification.
4) Deny or delay network access to the Distribution Service.

A cannot (by assumption):

- Break standard cryptographic primitives (digital signatures, SHA-256 hashes, authenticated encryption).
- Extract secret keys from the hardware enclave or the Model Owner.
- Compromise the integrity of the Distribution Service's authentication and access-control logic.

C. Trusted Computing Base (TCB)

Our TCB consists of:

- The hardware enclave used for training and MTD obfuscation, including its firmware and attestation chain.
- The Model Owner's private signing key and the correctness of the signature-generation process.
- The Distribution Service's storage and access-control mechanisms (but *not* its surrounding OS or network, which may be hostile).
- Standard cryptographic libraries and their implementations.

TABLE I
SUBSET OF PYTORCH MODEL ARCHITECTURES USED TO VALIDATE THE SUGGESTED MTD METHOD.

Model Type	Model Architecture	#Models	Size [MB]
Vision (Classification)	AlexNet [25]	1	230
	ConvNext [26]	4	110-750
	DenseNet [27]	4	30-110
	EfficientNet [28], [29]	12	20-450
	RegNet [30]	34	20-2460
Vision (Segmentation)	DeepLabV3 [31]	3	40-230
Vision (Detection)	Faster R-CNN [32]	4	70-160
Vision (Video)	SwinTransformer [33]	4	110-360
Text	BERT [34]	8	440-1440
Audio	Wav2Vec [35]	2	380-1270

D. Out-of-Scope Scenarios

The following are expressly excluded from this work:

- **Data poisoning, backdoor insertion during training, or adversarial-example attacks at inference time.** These alter model *behaviour* rather than the on-disk artefact and are orthogonal to file-level protection.
- **Full host compromise *before* MTD creation** (if the training environment is already under attacker control, any file- level defence is moot).

VI. DATASET

For evaluation of our suggested method, we create 2 datasets: benign AI models, and AI models attacked with steganography.

Dataset 1 - Benign PyTorch models: A dataset of benign pre-trained PyTorch models. We compile this dataset using the torchvision and HuggingFace APIs. Table I contains a subset of the pre-trained model architectures we used. In total, this dataset contains 129 vision, text, and audio models. We use this to validate our method on benign data, and it is used as a base for creating a dataset of models attacked with steganography (dataset 2) for further evaluation.

Dataset 2 - Attacked models for MTD method evaluation: We use LSB model weight steganography attacks [36] from our previous work on the benign models from dataset 1 to validate the MTD verification, ensuring that the MTD process works as intended with untempered models and also that the method successfully detects changes (i.e., physical attacks) in the model as intended.

VII. EVALUATION

This section evaluates our proposed AI model MTD method. The experiments are designed to prove that the method has practical value for protecting AI model ecosystems from physical attacks such as data exfiltration or unauthorized access to the trained AI model.

Experimental Setup: All experiments are run on a Ubuntu Linux 24.04 server, equipped with an Intel(R) Xeon(R) w5-2445 CPU (20 cores @ 3.10GHz), and 128 GB RAM.

Baseline Method: In terms of obfuscating the model to prevent unauthorized access, the immediate solution is to encrypt the serialized payload before transmission. This is also suggested by the NSA CSI [5] to ensure security of the model files. In our experiments, we compare our suggested MTD method to the Fernet encryption algorithm in terms of runtime. The MTD approach has the property of keeping the model file accessible, i.e., recognizable as the original model file, but the trained state is only available to the rightful owners of the model. This can be a benefit, for example, in scenarios such as model hubs that index the model and give an overview of its metadata, like HuggingFace does; with encryption, this won't be possible.

A. Experimental Results

We evaluate the MTD method for protecting training models with 2 main concerns: validity and runtime. Validity means we assert that the method prevents the attacks it's supposed to prevent and that the models stay intact after the whole process. We measure runtime to analyze the time overhead that the MTD methods cause. On the model sender side, there are the obfuscation and saving (serialization) methods, and on the model receiver side, there are the loading (deserialization) and the deobfuscation methods. We created automated Quality Assurance (QA) tests to check the method's validity. The tests check the following:

1) MTD model is the same after obfuscation and deobfuscation.
2) MTD model is the same after saving and loading.
3) Attack simulation: An MTD Model is constructed, obfuscated, and saved. Then the model weights are attacked, and we assert that the change is detected upon load.

For steps 1 and 2, we use the benign PyTorch models dataset (dataset 1, see Section VI); for step 3, we use dataset 2. All tests passed. Figure 4 plots the mean runtime of saving and loading MTD functions on all models in dataset 1, repeated 10 times. Additionally, we compare the runtimes to regular saving and loading to quantify the additional overhead our suggested method adds. Saving and loading were done with a virtual file since we are only concerned with the runtime overhead our method adds. We can see our MTD method adds approximately 4 seconds of runtime overhead to the serialization of a 2500 MB model; we feel this is within reason. Additionally, looking at Table II, we can see MTD obfuscation/deobfuscation has almost no runtime overhead, while the alternative method of encrypting the payload has a significant runtime overhead (about 17 seconds for a 2500 MB model). Therefore, in total, choosing our MTD method will be faster than using encryption. Finally, the MTD deserialization method has nearly identical runtime to regular runtime. In conclusion, the suggested MTD method is faster than encryption, especially as model size scales, and it also keeps the model file accessible in terms of metadata, structure, etc., which can be useful for sites like HuggingFace that give high-level overviews based on the model metadata.

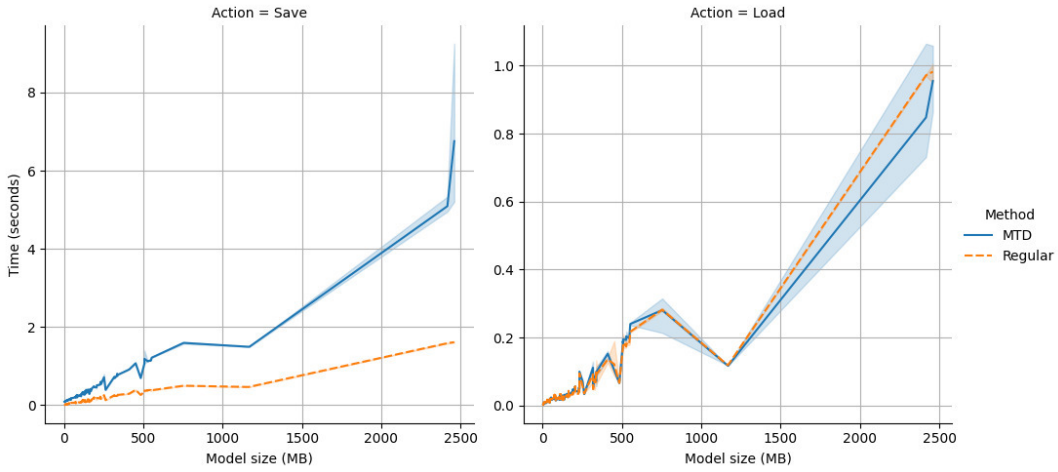

Fig. 4. Mean time (seconds) measurement vs. model size (MB) of saving/loading MTD actions. We show the MTD variant (solid blue line) and the regular variant (dashed orange line) for save and load. The action procedures were repeated 10 times with each model in the MTD datasets, and we show confidence intervals. We can see a linear increase in serialization (saving) runtime as the model size grows, and MTD has little to no overhead on loading the model after serialization.

TABLE II
RUNTIME (SECONDS) OF SERIALIZATION/DESERIALIZATION USING OUR SUGGESTED MTD METHOD, AND THE BASELINE FERNET ENCRYPTION METHOD.

	Model Serialization		Model Deserialization	
	MTD Obfuscate	Fernet Encrypt	MTD Deobfuscate	Fernet Decrypt
$0 \leq$ Model Size (MB) $<$ 100	**0.003820 ± 0.002000**	0.244292 ± 0.115333	**0.005795 ± 0.003956**	0.245123 ± 0.123005
100 $\leq$ Model Size (MB) $<$ 200	**0.008666 ± 0.002466**	0.680410 ± 0.126848	**0.013184 ± 0.005763**	0.724327 ± 0.133060
200 $\leq$ Model Size (MB) $<$ 300	**0.014490 ± 0.004417**	1.225329 ± 0.231984	**0.021955 ± 0.009834**	1.197840 ± 0.278559
300 $\leq$ Model Size (MB) $<$ 400	**0.017144 ± 0.002250**	1.648033 ± 0.388462	**0.025935 ± 0.002614**	1.746837 ± 0.373579
400 $\leq$ Model Size (MB) $<$ 500	**0.024434 ± 0.006219**	2.898500 ± 0.236313	**0.040424 ± 0.018690**	2.574974 ± 0.215552
500 $\leq$ Model Size (MB) $<$ 600	**0.019980 ± 0.001987**	3.002284 ± 0.217727	**0.021760 ± 0.004777**	2.574076 ± 0.170747
700 $\leq$ Model Size (MB) $<$ 800	**0.037661 ± 0.005977**	5.552645 ± 0.103848	**0.051598 ± 0.006328**	5.999168 ± 0.116049
1100 $\leq$ Model Size (MB) $<$ 1200	**0.051498 ± 0.000437**	7.697417 ± 0.152804	**0.069661 ± 0.002010**	8.562226 ± 0.158626
2400 $\leq$ Model Size (MB) $<$ 2500	**0.109626 ± 0.023599**	17.901734 ± 0.261984	**0.134600 ± 0.019062**	19.233927 ± 0.467096

VIII. LIMITATIONS

The proposed solution introduces a novel architecture for physically securing AI model files from malicious exposure. The architecture prevents all file alterations when the model is secured with the suggested MTD method. This builds upon previous work on CDR against steganography attacks [37] and steganalysis [36] (steganography attack detection). Since AI model file security is still in its early stages, we expect to see more sophisticated attacks and vulnerabilities in this domain. The proposed MTD architecture requires the user to fully use the various AI model functions, such as training and serialization within it, but widely-used methods like SafeTensors [38] do so too. Moreover, the proposed method adds a certain performance overhead that is added to all models since it works in a zero-trust manner. However, the performance overhead is small, as described in the results section. It is important to note that this work focuses solely on the security of the AI model's physical file. Other adversarial attacks exist, such as backdoors in machine learning models [1] and dataset poisoning attacks [39]. Dataset attacks are outside this paper's scope, but future work should view them as part of a holistic platform.

IX. CONCLUSIONS

Our work proposes a new solution for securing AI model files by preventing file modifications and unauthorized access through the use of MTD and authentication. This architecture aims to secure AI model file transfers from malicious attacks that threaten the industry. By incorporating MTD, we follow the NSA's [5] recommendation to separate the model architecture and its weights. This is accomplished by randomizing the model weights and separating the model from the deconstruction mapping. This ensures that only authenticated users can reconstruct and use the model when it is distributed. Based on our experimental results, the proposed architecture provides a 100% attack prevention rate (129/129 tampered models were correctly alerted) under the assumed conditions. In addition, we show that our method is significantly faster than using encryption (0.1 seconds to obfuscate vs. 18 seconds to encrypt

for a 2500 MB model), and contrary to encryption, it also keeps the model file in the original format, which can allow to get a high-level overview of the model, based on the metadata (e.g., number of parameters, etc.). Of course, users can also choose to use encryption on top of our suggested method if they wish to. Our method was evaluated on a wide range of data, which includes diverse benign models and steganography attacks from the literature and real-world scenarios found in HuggingFace [40]

X. Acknowledgement

This work is under US Provisional Patent Application No. 63/536,420

References

[1] E. Wenger, J. Passananti, A. N. Bhagoji, Y. Yao, H. Zheng, and B. Y. Zhao, "Backdoor attacks against deep learning systems in the physical world," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 6206–6215.

[2] M. Chen, G. He, and J. Wu, "Zddr: A zero-shot defender for adversarial samples detection and restoration," *IEEE Access*, 2024.

[3] K. Nguyen, T. Fernando, C. Fookes, and S. Sridharan, "Physical adversarial attacks for surveillance: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[4] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.

[5] N. S. Agency, "Deploying ai systems securely, best practices for deploying secure and resilient ai systems," 2024, accessed: 2024-05-01. [Online]. Available: https://media.defense.gov/2024/Apr/15/2003439257/-1/-1/0/CSI-DEPLOYING-AI-SYSTEMS-SECURELY.PDF

[6] M. ATLAS, "Mitre atlas," 2024, accessed: 2024-05-01. [Online]. Available: https://atlas.mitre.org/

[7] OWSAP, "Owasp machine learning security top ten," 2023, accessed: 2023-10-15. [Online]. Available: https://owasp.org/www-project-machine-learning-security-top-10/

[8] MITRE, "Mitre atlas," 2023, accessed: 2023-10-15. [Online]. Available: https://atlas.mitre.org/

[9] M. ATLAS, "Mitre atlas, user execution: Unsafe ml artifacts," 2024, accessed: 2024-05-01. [Online]. Available: https://atlas.mitre.org/techniques/AML.T0011.000

[10] R. Dubin, "Disarming attacks inside neural network models," *IEEE Access*, 2023.

[11] E. Sultanik, "Never a dill moment: Exploiting machine learning pickle files," 2022, accessed: 2022-12-19. [Online]. Available: https://blog.trailofbits.com/2021/03/15/never-a-dill-moment-exploiting-machine-learning-pickle-files/

[12] P. Zhou, "How to make hugging face to hug worms: Discovering and exploiting unsafe pickle.loads over pre-trained large model hubs," accessed: 2024-08-01. [Online]. Available: https://i.blackhat.com/Asia-24/Presentations/Asia-24-Zhou-HowtoMakeHuggingFace.pdf

[13] M. Slaviero, "Sour pickles, a serialized exploitation guide in one part," accessed: 2023-05-01. [Online]. Available: https://media.blackhat.com/bh-us-11/Slaviero/BH_US_11_Slaviero_Sour_Pickles_Slides.pdf

[14] D. Gilkarov, "AI-MTD Code Repository," 2025, accessed: 2025-06-04. [Online]. Available: https://github.com/ArielCyber/AI-model-MTD.git

[15] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.

[16] V. Heydari, "Moving target defense for securing scada communications," *IEEE Access*, vol. 6, pp. 33 329–33 343, 2018.

[17] M. Azab and M. Eltoweissy, "Migrate: Towards a lightweight moving-target defense against cloud side-channels," in *2016 IEEE security and privacy workshops (SPW)*. IEEE, 2016, pp. 96–103.

[18] M. Styugin, V. Zolotarev, A. Prokhorov, and R. Gorbil, "New approach to software code diversification in interpreted languages based on the moving target technology," in *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*. IEEE, 2016, pp. 1–5.

[19] S. Banescu and A. Pretschner, "A tutorial on software obfuscation," *Advances in Computers*, vol. 108, pp. 283–353, 2018.

[20] G. Mordehai, Y. Elovici, and G. Kedma, "Method and system for protecting computerized systems from malicious code," Jul. 11 2017, uS Patent 9,703,954.

[21] D. Evans, A. Nguyen-Tuong, and J. Knight, "Effectiveness of moving target defenses," *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, pp. 29–48, 2011.

[22] R. Dubin, "Disarming attacks inside neural network models code repository," 2022, accessed: 2023-07-15. [Online]. Available: https://github.com/ArielCyber/AI-MODEL-CDR

[23] Intel, "Reference architecture for privacy preserving machine learning with intel® sgx and tensorflow* serving," accessed: 2023-05-01. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/privacy-preserving-ml-with-sgx-and-tensorflow.html

[24] "The ai pc powered by intel is here. now, ai is for everyone." accessed: 2023-05-01. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/processors/core-ultra/ai-pc.html

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[26] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," 2022. [Online]. Available: https://arxiv.org/abs/2201.03545

[27] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018. [Online]. Available: https://arxiv.org/abs/1608.06993

[28] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020. [Online]. Available: https://arxiv.org/abs/1905.11946

[29] ——, "Efficientnetv2: Smaller models and faster training," 2021. [Online]. Available: https://arxiv.org/abs/2104.00298

[30] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," 2020. [Online]. Available: https://arxiv.org/abs/2003.13678

[31] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017. [Online]. Available: https://arxiv.org/abs/1706.05587

[32] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016. [Online]. Available: https://arxiv.org/abs/1506.01497

[33] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video swin transformer," 2021. [Online]. Available: https://arxiv.org/abs/2106.13230

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[35] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," 2020. [Online]. Available: https://arxiv.org/abs/2006.11477

[36] D. Gilkarov and R. Dubin, "Steganalysis of ai models lsb attacks," *IEEE Transactions on Information Forensics and Security*, 2024.

[37] R. Dubin, "Content disarm and reconstruction of steganography malware in neural network models," 2023, accessed: 2022-01-15. [Online]. Available: https://github.com/randubin/CDR-NN

[38] H. Face, "Safetensors," accessed: 2024-08-01. [Online]. Available: https://huggingface.co/docs/safetensors/en/index

[39] J. Zheng, P. P. Chan, H. Chi, and Z. He, "A concealed poisoning attack to reduce deep neural networks' robustness against adversarial samples," *Information Sciences*, vol. 615, pp. 758–773, 2022.

[40] H. Face, "Hugging face model zoo," accessed: 2023-05-01. [Online]. Available: https://huggingface.co/models