Evolving Malware: Using AI to Outsmart Static and Dynamic Analysis

Flávio Lobo Vaz
Faculty of Engineering of Porto.
University of Porto
Email: up201509918@fe.up.pt

Pedro Rezende de Carvalho
Faculty of Engineering of Porto
University of Porto
Email: up201900513@fe.up.pt

***Abstract*—AI has been used to improve a multitude of tools, for malware evasion is no different, in this paper we summarized some of the state-of-art tools for AI-based malware generation and propose a new framework that combines some of these tools to improve on their results. With the combined power, using malware samples from a dataset of VirusTotal, we could increase the number of miss classifications as benign to about 42.5% and also achieve a score of 12 / 71 in VirusTotal. While retaining the malicious functionality of the original malware.**

I. Introduction

Artificial Intelligence (AI) have been implemented in a multitude of tools and context, from text generation to image processing. Not differently, malware detection systems are taking advantage of AI to improve. By combing with traditional methods, anti malware systems can achieve even better results, been today de standard of most commercial antivirus[4].

As the detection system evolves, so does the evading techniques. AI has been known to have vulnerabilities, a common one is adding noise to images can alter completely the result of an image classifier[7]. In the malware generation context, some frameworks that use AI have been proposed, [10][36]. Each one applying different techniques to generate evasive malware.

In our work, we propose a new framework that joins some of the state-of-art AI-based frameworks, to get an improvement on the results. We had a primary focus on evading static analysis and, as secondary, dynamic analysis.

II. Related work

A. Technical Background

1) Malware: The term malware is short for of malicious software, and apply to any kind of software that intended to harm computers and computer users by stealing information, corrupting files or by just doing mischievous activities, some of the forms they can take are: Virus, Worms, Trojan Horses, Spyware, Adware, Sniffers, Keyloggers, Ransomware, etc.[5]

2) Traditional Malware Evasion Techniques: Evading techniques, mainly target to avoid being fully analyzed or being detected, can be classified as anti-analysis and obfuscating techniques[6]. Obfuscating techniques can apply when writing malware to change the syntax without changing the semantics, some common strategies are[6]:

- Dead-Code Insertion, where useless code is inserted in the code body.
- Code Transposition, is the shuffle of the binary order to change the order of execution.
- Register Reassignment, which uses the replacement of register.
- Instruction Substitution, rewriting code to substitute an existing one.

The resulting malware can also apply some obfuscating techniques at runtime and can be classified in the following types: encrypted, oligomorphic, polymorphic and metamorphic.[9]

3) AI Based Evasion Techniques: AI have been used to improve on non-AI strategies. Some strategies focus on improving the mutation generator of polymorphic and metamorphic malware, while other focus on the creation of the malware itself[8]. Some frameworks have been created using machine learning, they focus on using a combination of obfuscating techniques trying to find the best ones for a particular discriminator, usually Anti-virus classifiers that commonly use machine learning[10].

4) Windows PE files: PE stands for Personal Executable, it is a very common file format for windows operating systems. Commonly used by Executables (".exe") and Dynamic Linked Library (".dll"), used on both 32 and 64 bit systems. The PE file system compose of a series of headers, containing information about the machine and code, a section with the text, the actual instructions, to be executed, and finally a data section[15].

5) GAN: The foundation of Generative Adversarial Networks is a game, in the sense of game theory, between two machine learning models, which are typically neural networks. The Generative Adversarial Network (GAN) is an algorithm for machine learning consisting of two distinct deep learning networks: Discriminator and Generator. They compete to win a zero-sum game[3]. If the Discriminator network is accurate, feedback is sent to the Generator network, so it can modify its weights and probability distributions in order to improve the quality of its forgeries. If the Discriminator makes a mistake, this information is sent back to it, in order that it can make the necessary adjustments[1]. When the Discriminator has the accuracy of a coin toss or, in other words, when it is virtually impossible to distinguish forgeries from the Generator from

authentic samples, the competition ends. The Generator has "won" when the Discriminator obtains a success rate of 50%. GANs are, at their core, generative models that can generate new samples from a provided dataset[2].

6) Reinforcement learning: Reinforcement learning (RL) is a subfield of artificial intelligence with a set of techniques, predominantly focused on how an agent should act in an environment in order to optimize some concept of cumulative reward. It is particularly suitable for problems involving a sequential decision-making approach[28]. In brief, words, an agent interacts with its environment in discrete time steps. At each time step, the current environment's state is checked by the agent which chooses an action. In response, the environment will enter a new state, and the agent receives a reward. Thereby, the purpose of the agent is to find a policy, that represents a mapping between states and actions, which optimizes the expected cumulative reward [17]. It is also important to note, that currently Deep Reinforcement Learning (DRL), the combination of RL and deep learning, has recently achieved significant success, allowing previously intractable complex problems to be handled[16].In a sentence, in contrast with traditional deep learning, RL permits the influence of past actions on future actions[28].

7) Antivirus: Are programs that are able to scan and detect malicious code, some methods commonly used by antivirus are[8]:

a) Signature Based Method: In this technique the antivirus look for sequence of code or data that match a pattern, called signature, in their database, it has a small error rate, but it cannot detect variants or new malware and also require a large database[8].

b) Behavior based Method: Instead of looking at the malware itself, this approach focus on what the malware does and look for uncommon behavior, it also uses signatures but for behavior instead of code or data. This strategy can have many false positive and thous require a lot of scanning[8].

c) Heuristic based method: Instead of applying pattern matching and signatures, this strategy applies heuristics to decide if the malware is or not malicious, it usually examines the normal behavior to determine what is abnormal. The main advantage is that variants can easily be detected and even new malware, given the heuristics knows the vulnerability. Some drawbacks are the heavy computational load and can have a high false positive rate[8]

8) Malware machine learning classifiers: To improve classification, Machine Learning have been adopted in malware detection software. Some common technologies applied are data mining and Neural Networks, that aim to analyze malware and detect similarities, somewhat similar to heuristic based detection but letting the model figure what is or not malicious based on the dataset, they can even use sandboxing, together with dynamic analysis, to look at the interactions to better classify a file[8].

B. Application of Technical Concepts in Our Project

1) Used Malware Evasion Techniques: Most of the techniques used are for obfuscating the code, making it look and behave more link a benign software, some of them include[10][19]:

- Adding imports and DLLs.
- Adding and renaming sections.
- Append to code existing sections.
- Adding and removing signatures.
- UPX Pack/Unpack.
- Changing header information; removing debug flags, certificates, checksums, etc.

2) Used tools to parse, manipulate and reconstruct PE32 files: Some tools can be used to alter the content of PE file, such as LIEF[12], a library that allow us to parse and modify PE, Mach-O and ELF files, PE Bliss[13], a C++ library useful for rebuilding PE files, and UPX[14], a tool for packing and unpacking executables.

3) MAB-Malware: Mab-Malware is a framework base on Reinforcement Learning (RL) that aims to improve on the traditional methods of applying RL for generating malware. The common method use a policy with mutations to find the best combination of features that can evade a detection system.

The first technique used to improve traditional RL implementations, it is the way MAB-Malware treats mutations. Traditional methods consider mutations in a stateful way, meaning that the actions depend on one another, this increases a lot the search space, since we need to consider the combination of mutations. MAB-Malware treats each mutation as independent and so apply a stateless strategy, this reduces the search space, reducing the learning difficulty[10]. With this strategy MAB-Malware model the problem into a Multi-armed-bandit (MAB) problem. The problem can be visualized as a gambler in front of n slots machines and has to maximize it gains by exploiting the machines and choose when to change to a different machine[11], in this formulation the slots machine can be seen as the mutations to be applied to the resulting malware.

The second improvement is on the content that is inserted in the malware. In traditional RL methods, if the action to be taken is, for example, an add new section the content of that section is picked randomly. MAB-Malware consider the content as an important part for success of the resulting malware, so because of this it doesn't treat the actions alone but in an action-content pair[10].

Lastly, when the malware is successfully created the policy needs to be updated to take into account which action-content pair works. In normal RL all the actions used on the successful malware would be rewarded, allowing actions that didn't affect the final results to be prioritized. MAB-malware uses a minimizer to figure which action-content pair actually was necessary, and only those would be rewarded, this allows the framework to be more efficient, by removing confusing rewards, as well as figure what was the root cause of the evasion[10]. The resulting workflow can be seen in figure 1.

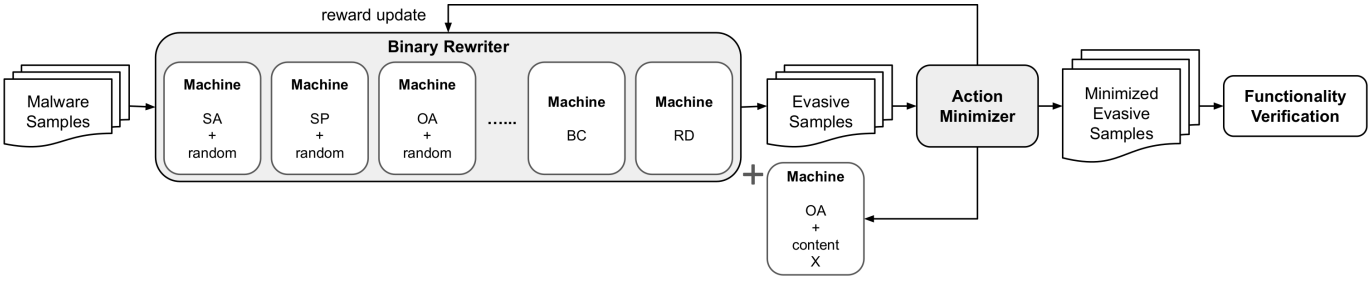

Fig. 1. MAB-Malware workflow [10]

In the original paper the framework used 5000 PE malware that were all detected by VirusTotal[22]. As a result, comparing with over off-the-shell frameworks, MAB-Malware was able to get an increment of 23% on detection on state-of-the-art ML detectors and an increment of 16% on commercial Antivirus[10].

4) Pesidious: **Pesidious** is a framework that leverages a combination of deep reinforcement learning (DRL) and Generative Adversarial Networks (GAN) to mutate malware samples, particularly PE32 files. Therefore, the objective is to evade anti-malware machine learning classifiers, while keeping their original functionality[20].

The framework primarily consists of five sequential phases, parsing, feature extraction, malware mutation, rebuilding the malware and classification, more information can be found here[48].

The **results** of Pesidious are somewhat ambiguous. What was possible to find out by watching an online presentation[38] having access to the little information[20] available was that: after a lot of trial and error, the mutated malware in general, continued functional, only 80 mutations or less can be done. The Bliss (written in C++) tool was also crucial to maintain the functionality.

As an example, the researchers, in the end, claim that the best results achieved on VirusTotal were lowered from a score of 66/67 to 37/67, making the muted malware approximately 43% more evasive than its original version, with full functionality,

A figure that summarizes Pesidious can be found here[47]

5) Quo-vadis classifier: It consists of a hybrid architecture that uses three distinct analysis techniques: a Windows emulator to acquire behavioral patterns (expressed as a sequence of Windows kernel API calls) across large corpora, contextual information in the form of a file path on a system at the moment of execution of the malware and static representations obtained from the Windows PE structure (Ember classifier[32]). Minimal temporal and computational costs were achieved[45].

It uses a large dataset acquired with the collaboration with a security vendor, having over one hundred thousand in-the-wild malware samples.

The hybrid classifier (using the above three distinct analysis techniques) surpasses the current state-of-art models in terms of detection rates, particularly with minimal false-positive requirements (benign programs that are classified as malware) like 0.25%, where all scores F1-score, Precision, Recall, Accuracy, and AUC [33][34] on all sets were reported with decision threshold of 0.98 on the validation set. In the out-of-sample test set, F1 and AUC scores decrease scores by approximately 4-4.5%[45].

III. Contribution

Given all that was mentioned in the related work and Application of Technical Concepts in Our Project, the idea of a new framework was born, **NeoAthena**. The primary contribution of this project is that NeoAthena combines the strengths of two existing frameworks, **MAB** and **Pesidious**, to improve malware mutation and evasion against antivirus and anti-malware machine learning classifiers. Other contributions, which in this work have less weight, where their automation and total integration with NeoAthena are left for future work, will also be mentioned, but for a more detailed explanation, a link comparing results with the use of MAB and Pesidious against also using the other contributions as well as good references are given.

A. Methodology

This work combines MAB and Pesidious with the goal of achieving new malware evasion outcomes. The quo-vadis classifier has been integrated to provide a comparison classifier that takes into account aspects beyond static analysis. MalDiv and MalWash have also been incorporated, but are not yet fully integrated and automated and for now just used for comparisons. TheFatRat and Shellter have been used to create a malware sample and also to compare with the rest created by NeoAthena using MAB and Pesidious. The ultimate goal in the present and the future of NeoAthena is to allow the full, relatively quick, and organized integration of new types of algorithms GAN, DRL among other techniques that allow making malware more evasive against static and dynamic malware analysis from machine learning classifiers and antivirus.

1) Addressing the problem: First, we had to decide in what direction the mutations would have to be carried out, that is, in which direction they would communicate. After a careful analysis, we realized that the fact that Pesidious used as possible mutations UPX pack/unpack, the MAB would not have a way to use the malware samples mutated firsthand by the Pesidious, as it has no way to deal with PE files that had suffered UPX packs. Thus, the NeoAthena framework

in relation to mutations and the way they communicate will always be in the MAB - Pesidious direction.

By carefully analyzing the MAB and Pesidious code [41] [36], we found that the Pesidious framework is based on Python 3.6, which is currently not supported. This would make it very difficult to fully integrate with MAB, quo-vadis, and other frameworks. For a matter of time management, we used the following workaround:

- Pesidious, quo-vadis, and MAB among other frameworks are all installed in the same directory.
- So, as the MAB runs using a docker image and the quo-vadis is relatively easy to use, when we want to do something more specific with compatibility issues outside the Pesidious or using other frameworks, we use the main directory, otherwise, in general, even importing muted malware by MAB we manage to do from within the Pesidious.

Conflicts related to Python 3.6 and directory manipulation problems from the docker that runs the MAB have been solved to be easily accessible from the outside. Some of the Pesidious and quo-vadis code has been modified (mainly code related to classifiers and mutations), as well as scripts have been developed to make the entire process of malware mutation of the final NeoAthena framework almost fully automated.

Something important to highlight, is that in order to not perform unnecessary mutations and risk loss of original malicious functionality, after the mutations made by the MAB, mutations involving the Pesidious will only be performed to the mutated malware that after passing through MAB is still classified by our classifier as malware.

To separate the mutated malware either only by the MAB or MAB and Pesidious together, we have modified the Pesidious and used a classifier to save in different directories the samples of mutated malware that are classified as benign (and therefore evade the classifiers) or still classified has malware.

All malware samples used in MAB and Pesidious came from VirusTotal, from the dataset used in MAB. Out of a total of 1000, approximately 800 are mutated by MAB and sent to Pesidious. As the samples came from VirusTotal, for simplicity, we assume that they are all recognized by almost all antiviruses in VirusTotal [10].

All the flow of the NeoAthena framework is made using a menu that has been implemented and automated by us, which can be seen in the figure below as well as in more detail in references to the repository. The main script to run the flow and menus is run NeoAthena.py which is located inside Pesidious.

Thus, the final flow of the NeoAthena Framework after being functional can be seen in figure 2.

2) Design of the Architecture: The final design can be seen in figure 3.

B. Findings

1) Limitations: Much due to computing power limitations (a PC with 16 GB RAM was used, with Processor Intel® CoreTM i7-1065G7 CPU @ 1.30GHz × 8) the MAB mutation process of 800 malware samples took around 3 hours. Mutations from just 15 to 20 samples of malware already mutated by MAB on Pesidious can take more than an hour.

So, because we wanted to have tangible results for the presentation, we decided to go through the entire flow of NeoAthena using 40 randomly collected samples from over 800 samples of malware already mutated by the MAB.

The use of the quo-vadis classifier was limited to its use in two samples of mutant malware that were used in the presentation and that were chosen by us due to their score given by the gradient-boosted classifier. One of the difficulties found is that in order to use the hybrid model to its full potential, we have to give the supposed path of a directory where the possible malware was supposedly founded running. For this, we decided to give a path in the format /User/Downloads simulating the used context taken into account by us: a simple download on a local network of a sample of malware. It is also important to note that using this allows us to have a layer of greater certainty regarding the functionality of the mutated malware. Since the API calls are simulated. However, we cannot fully trust this fact, as we discovered that there may be errors due to the fact that the Windows Kernel emulator used does not support some types of API used by the malware.[45]

2) Results overview: The Pesidious framework, despite the immense limitations due to problems with Python 3.6 compatibility and lack of documentation of the source code, after an extensive analysis, proved to be relatively well organized to accumulate new mutations from other frameworks.

As already described, the use of MalDiv, MalWash as well as FatTheRat with Shellter, also for lack of time, ended up having a limited integration and only used in a sample of Malware in terms of comparison with what was generated in NeoAthena only with MAB and Pesidious. The results and brief description of the tools will be in the Readme of the repository that will be provided.

Overall, we have been successful in generating better results with the NeoAthena framework than the use of MAB and Pesidious used separately, evading the classifier or on VirusTotal. The concrete results will then be presented in the assessment.

We generally assume the functionality of the malware, backed by the results of Pesidious and MAB, in the use of the quo-vadis Windows kernel emulation (with some limitations here) but also the utilization of two virtual machines, with Cloud-Based Secure Sandboxed Environment running dynamic analysis on the best two samples used on the presentation given about NeoAthena.

C. Significance

In general, the NeoAthena framework uses state-of-the-art techniques, frameworks, and classifiers, trying to make all this as integrated and modeled as possible (both in present and in future work) so that more features can be added whenever desired. It achieves relevant results, either in the evasion of the classifier used by us, mainly considering that the results obtained do not select any family of malware in particular and compared with the results of Pesidous and MAB when used

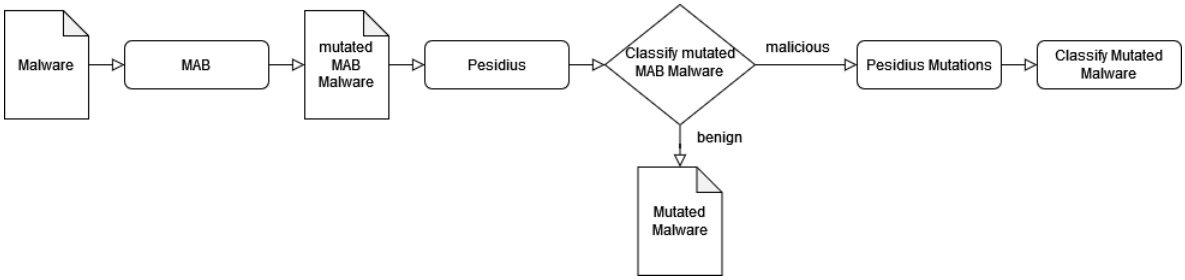

Fig. 2. NeoAthena final flowchart

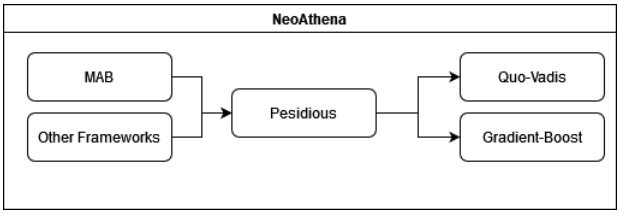

Fig. 3. NeoAthena workflow

separately. Also, compared with the results of the developers of Pesidious on VirusTotal, our best mutated malware sample (with a score given of approximately 0.1 by the gradient-boost classifier), achieves a better score than what was demonstrated by Pesidious developers. This shows the relative transferability of the fact that we successfully evade our classifier, this transfers for a series of antivirus in VirusTotal that possibly uses machine learning classifiers.

It is also important to note, that the mutations made by NeoAthena, are possible to observe in the field details of the report given by VirusTotal when a malware sample is submitted.

IV. Assessment

A. Results

1) Evading the gradient-boost classifier and VirusTotal using the NeoAthena framework :

1) Out of 40 MAB-mutated malware samples sorted by gradient boost:
   a) 35% are classified as benign: 14 out of 40
   b) 65% is rated by malware: 26 out of 40
2) After the process with the NeoAthena framework (MAB and Pesidious mutations)
   a) 77.5% evade the gradient-boost classifier
   b) 22.5% is classified as malware
3) The best sample has a score of 12 / 71 in VirusTotal (MAB and Pesidious mutations)
4) Link for VirusTotal best sample submission [40]
5) Link for the cloud-based dynamic analysis[42]
6) The above cloud-based dynamic analysis doesn't detect any malicious behavior.
7) The demo samples (some of our best ones) are functional.
8) Results about the use of the other frameworks to compare are on the README of the repository, the samples used for the results above are in a directory identified by **results** in the repository. As other important information that was not possible to submit here.
9) Repository link [47]
10) Results about the use of the other frameworks to compare are on the README of the repository, the samples used for the results above are in a directory identified by **results** in the repository. As other important information that was not possible to submit here.

V. Conclusion

In general, the NeoAthena framework obtains promising results, better than using only MAB or Pesidious in isolation and without focusing on a specific malware family, as in other works[29]. We think that in the future, a more careful analysis may even find more concepts and techniques to improve the framework. The fact that we obtained a score of 12 / 71 on VirusTotal, having mutated malware samples from a dataset provided by VirusTotal, is an interesting result, that the developers of Pesidious could not achieve. After the NeoAthena flow, we obtained an evasion of the gradient-boost classifier of 77.5%, also a good result.

A. Future work

- Improve the work done so far, managing to use at least all 800 malware samples to get better and more robust results, but also improve integration of everything used.
- Reformulate the Pesidious framework, so that it can serve as a basis to fully and easily integrate other frameworks and improve NeoAthena.
- Fully integrate and improve: MalDiv, MalWASH, TheFatRat, and Shellter with NeoAthena.[47]
- Use another type of GAN, such as MalFox, or TransGan (use of transformers on GANs) [43] [50]
- Use deep reinforcement learning algorithms like self Play.[49]
- Study countermeasures to mitigate what was developed at NeoAthena.

References

[1] Goodfellow, Ian, et al. "Generative adversarial networks." Communications of the ACM 63.11 (2020): 139-144.

[2] Dunmore, Aeryn et al. “Generative Adversarial Networks for Malware Detection: a Survey.” ArXiv abs/2302.08558 (2023): n. pag.

[3] Roy, Sankardas et al. “A Survey of Game Theory as Applied to Network Security.” 2010 43rd Hawaii International Conference on System Sciences (2010): 1-10.

[4] Heena, Advances In Malware Detection- An Overview. 2021.

[5] TAHIR, Rabia. A study on malware and malware detection techniques. International Journal of Education and Management Engineering, 2018, 8.2: 20.

[6] A. P. Namanya, A. Cullen, I. U. Awan and J. P. Disso, ”The World of Malware: An Overview,” 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), Barcelona, Spain, 2018, pp. 420-427, doi: 10.1109/FiCloud.2018.00067.

[7] Comiter, Marcus . “Attacking Artificial Intelligence: AI’s Security Vulnerability and What Policymakers Can Do About It.” Paper, Belfer Center for Science and International Affairs, Harvard Kennedy School, August 2019.

[8] Jhonattan J. Barriga A. and Sang Guun Yoo ”Malware Detection and Evasion with Machine Learning” International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7207-7214 © Research India Publications. http://www.ripublication.com

[9] I. You and K. Yim, ”Malware Obfuscation Techniques: A Brief Survey,” 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, Fukuoka, Japan, 2010, pp. 297-300, doi: 10.1109/BWCCA.2010.85.

[10] Wei Song, Xuezixiang Li, Sadia Afroz, Deepali Garg, Dmitry Kuznetsov, and Heng Yin. 2022. MAB-Malware: A Reinforcement Learning Framework for Blackbox Generation of Adversarial Malware. In Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (ASIA CCS ’22). Association for Computing Machinery, New York, NY, USA, 990–1003. https://doi.org/10.1145/3488932.3497768

[11] Richard Weber ”On the Gittins Index for Multiarmed Bandits,” The Annals of Applied Probability, Ann. Appl. Probab. 2(4), 1024-1033, (November, 1992)

[12] Lief [n.d.]. LIEF. https://github.com/lief-project/LIEF.

[13] PE Bliss. https://github.com/BackupGGCode/portable-executable-library

[14] UPX - The Ultimate Packer for eXecutables. https://github.com/upx/upx

[15] M. Kozák and M. Jureček, Combining Generators of Adversarial Malware Examples to Increase Evasion Rate. 2023.

[16] Arulkumaran, Kai et al. “Deep Reinforcement Learning: A Brief Survey.” IEEE Signal Processing Magazine 34 (2017): 26-38.

[17] https://link.springer.com/content/pdf/10.1007/978-3-642-27645-3.pdf

[18]

[19] https://conference.hitb.org/hitblockdown002/materials/D2T2

[20] https://conference.hitb.org/hitblockdown002/materials/D2T2

[21] https://arxiv.org/pdf/1702.05983.pdf

[22] https://www.virustotal.com/gui/home/upload

[23] https://github.com/ZaydH/MalwareGAN

[24] Huang, Alex et al. “Adversarial Deep Learning for Robust Detection of Binary Encoded Malware.” 2018 IEEE Security and Privacy Workshops (SPW) (2018): 76-82.

[25] https://github.com/ZaydH/MalwareGAN/blob/master/malware_gan_poster.pdf

[26] https://github.com/ZaydH/MalwareGAN/blob/master/malware_gan_report.pdf

[27] Mnih, Volodymyr et al. “Human-level control through deep reinforcement learning.” Nature 518 (2015): 529-533.

[28] https://d2l.ai/chapter_generative-adversarial-networks/gan.html

[29] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou and H. Huang, ”Evading Anti-Malware Engines With Deep Reinforcement Learning,” in IEEE Access, vol. 7, pp. 48867-48879, 2019, doi: 10.1109/ACCESS.2019.2908033.

[30] Anderson, H. et al. “Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning.” ArXiv abs/1801.08917 (2018): n. pag.

[31] https://github.com/endgameinc/gym-malware/

[32] Hyrum S. Anderson and Phil Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv:1804.04637 [cs.CR]

[33] Powers, David M. W.. “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.” ArXiv abs/2010.16061 (2011): n. pag.

[34] https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/

[35] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[36] https://github.com/CyberForce/Pesidious/tree/master

[37] https://vaya97chandni.gitbook.io/pesidious/

[38] https://www.youtube.com/watch?v=aplHu2YMEFo

[39] Hu, Weiwei and Ying Tan. “Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN.” ArXiv abs/1702.05983 (2017): n. pag.

[40] https://www.virustotal.com/gui/file/3d63c59fc80b92ba67a8a5b2d3c1effa8ab5f70a224be

[41] https://github.com/bitsecurerlab/MAB-malware

[42] https://app.any.run/tasks/d1352f87-cbdb-465d-b635-77ba162b1edb/

[43] Jiang, Yifan et al. “TransGAN: Two Transformers Can Make One Strong GAN.” ArXiv abs/2102.07074 (2021): n. pag.

[44] Dhariwal, Prafulla and Alex Nichol. “Diffusion Models Beat GANs on Image Synthesis.” ArXiv abs/2105.05233 (2021): n. pag.

[45] Trizna, Dmitrijs. “Quo Vadis: Hybrid Machine Learning Meta-Model Based on Contextual and Behavioral Malware Representations.” Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security (2022): n. pag.

[46] Zhang, Lan et al. “Semantics-Preserving Reinforcement Learning Attack Against Graph Neural Networks for Malware Detection.” IEEE Transactions on Dependable and Secure Computing 20 (2020): 1390-1402.

[47] https://github.com/wolfCuanhamaRWS/NeoAthena-framework

[48] https://github.com/wolfCuanhamaRWS/Network-Security/blob/main/docs/Pesidious

[49] Bai, Yu and Chi Jin. “Provable Self-Play Algorithms for Competitive Reinforcement Learning.” ArXiv abs/2002.04017 (2020): n. pag.

[50] Zhong, Fangtian et al. “MalFox: Camouflaged Adversarial Malware Example Generation Based on C-GANs Against Black-Box Detectors.” ArXiv abs/2011.01509 (2020): n. pag.