Exploiting Windows PE Structure for Adversarial Malware Evasion Attacks

Kshitiz Aryal
karyal42@tntech.edu
Tennessee Tech University
Cookeville, Tennessee, USA

Maanak Gupta
mgupta@tntech.edu
Tennessee Tech University
Cookeville, Tennessee, USA

Mahmoud Abdelsalam
mabdelsalam1@ncat.edu
North Carolina A&T State University
Greensboro, North Carolina, USA

ABSTRACT

The last decade has seen phenomenal growth in the application of machine learning. At this point, it won't be wrong to claim that most technological change is directly or indirectly connected to machine learning. Along with machine learning, cyber-attacks have also bloomed in this period. Machine learning has been a great aid to cybersecurity, but the security of machine learning has not been a topic of attention until recently. Among numerous threats posed to the machine learning community, the Adversarial Evasion attack is the latest menace. The adversarial evasion attack has exposed the vulnerability of the modern deep neural network to a few intentionally perturbed data samples. The adversarial evasion attacks originated from the image domain but have now spread across major application domains of machine learning. This work will discuss the state-of-art adversarial evasion attacks against the Windows PE Malware detectors. The structure of a file plays a significant role in how an adversarial evasion attack can be carried out to a file. We will discuss the robustness and weakness of the Windows PE file structure toward the adversarial evasion approach. We will present the existing approaches to exploiting Windows PE file structure and their limitations. We will also propose a noble way to manipulate Windows PE structure to carry out an adversarial evasion attack.

CCS CONCEPTS

• Security and privacy → Malware and its mitigation.

KEYWORDS

Adversarial Evasion Attack, Windows PE Structure, Windows Malware, Malware Detector, Machine Learning

ACM Reference Format:
Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. 2023. Exploiting Windows PE Structure for Adversarial Malware Evasion Attacks. In Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy (CODASPY '23), April 24–26, 2023, Charlotte, NC, USA. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3577923.3585044

1 INTRODUCTION

Machine learning is thriving and automating many technologies that required lots of manual human intervention in the past. The phenomenal growth of machine learning has revolutionized modern technology, and the achievements of machine learning are driving major advances. With increased reliability and widespread adoption, machine-learning models are attracting the attacker recently in the ongoing cyber war. The latest threat to machine learning has arrived in the form of adversarial evasion attacks where small well-directed synthetic perturbations can thwart away the advanced machine learning architectures. The first few works of adversarial evasion attacks [3, 5] explored the vulnerability of machine learning models by adding a few imperceptible perturbations to an image. Adding perturbations to the image was easy as the only constraint was to make it imperceptible to the human eye. However, as the adversarial evasion attacks shifted to other domains, the domain-specific constraints limited the flexibility in carrying out the attack.

The adversarial attack on a malware detector is made by attacking a malware file before passing it to a malware detector. The malware needs to be recognized as a benign file by a detector while preserving the functionality and executability of a file. Recent works have made some progress in carrying out the adversarial attack in a malware domain [1, 2]. The crafting of adversarial malware examples is completely a different game than in the image. The strict semantic requirement of binary files can easily break the file even with minimal changes in the file. The adversarial evasion attacks in malware have always faced the trade-off between eficiency and practicality. More eficient attacks have lost practical value, while practical attacks are not suficiently eficient. Initial attacks on the malware file were made in feature space, where the practicality of attacks was overlooked [1]. The feature space attacks can be generated by crafting adversarial noise in the feature space. However, the malware features with this noise can not be mapped back to an executable file. Due to this limitation, recent advances in adversarial evasion attack has focused on problem space attacks.

The adversarial examples are generated by injecting a random ineffectual perturbation inside malware which will eventually help to bypass the malware detector. Random perturbations are an inefficient approach to attack; thus, different techniques are used to generate perturbations. The most common approach is by taking the gradient of a model under attack. The evolving attacks use more advanced approaches like reinforcement learning, GAN, and deep neural nets to generate perturbations. Even if the optimization algorithm has improved to a great extent, the process of perturbation injection still limits the adversarial evasion attacks in malware. In this work, different adversarial optimization algorithms are out of scope, and we will focus on the Windows PE file structure and its challenges. As we discussed earlier, the biggest challenge is to preserve the file from breaking while injecting these perturbations. We will discuss the structure of Windows PE file format and its vulnerabilities to adversarial evasion attacks. We will discuss how

CODASPY '23, April 24–26, 2023, Charlotte, NC, USA

ACM ISBN 979-8-4007-0067-5/23/04.
https://doi.org/10.1145/3577923.3585044

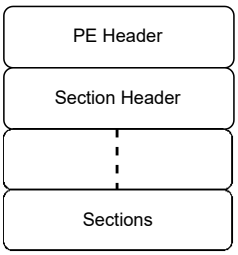

Figure 1: Basic Structure of Windows PE File

adversarial evasion attacks are performed by exploiting the structure of Windows PE files to inject perturbations. While discussing the limitations of existing work, we will also propose a novel way to manipulate Windows PE structure to insert adversarial perturbations for malware evasion.

2 BACKGROUND

2.1 Malware Detection

Malware detection is broadly classified into static, and dynamic detection [6]. Static detection deals with reverse engineering a file and feature analysis to detect malware without executing it. While dynamic analysis is the approach where malware is executed in an isolated environment, and based on the behaviors of the malware, the detection is done. All of the adversarial attacks in malware in done against static detection techniques, as the goal is to preserve the behavior of malware. The recent adversarial evasion attacks are mostly against end-to-end malware detectors like MalConv [9].

2.2 Windows PE File Structure

Windows PE file format is an executable file format based on the Common Object File Format (COFF) specification. The basic structure of Windows PE is shown in Figure 1. The PE header comprises the MS-DOS MZ header, the MS-DOS stub program, the PE file signature, the COFF file header, and an optional header. The PE header is one of the most sensitive regions in a file where small modifications can entirely break the file. While saying this, there are regions like an optional header that can be used to inject perturbations. Section headers contain information like Name, physical address, virtual address, size of raw data, a pointer to raw data, and different characteristics flags. Any modification in the section requires the section headers to be modified accordingly. The sections are the main contents of the PE file, which have different sections for various data stored. The executable codes are placed inside the .data section; uninitialized codes are inside the .bss section; the .rdata section contains read-only data like constants, and the .rsrc section contains resource information.

3 ADVERSARIAL EVASION ATTACKS ON WINDOWS PE FILE

Different attacks over time have exploited different regions of Windows PE files to perform an adversarial evasion attack. The most common approach has been by appending the perturbations at the end of PE file [7, 8] as shown in the left of Figure 2. There is a unique advantage of appending perturbations at the end, as no alterations are made to the functionality of a file. The end regions

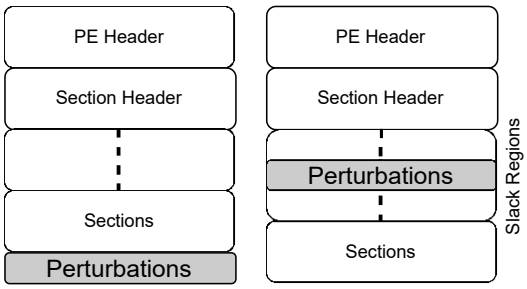

Figure 2: Append attack and Slack attack

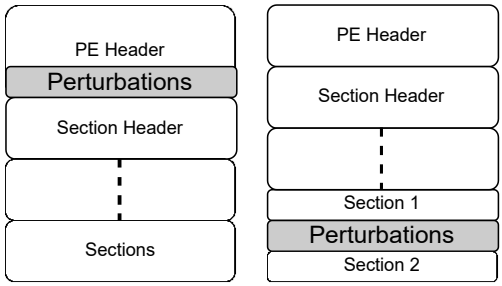

Figure 3: Header attack and Inserting Code Caves

are outside the header description and are never executed during the file execution. However, there are certain demerits of append attacks. Appending at the end will increase the size of a file while making perturbations easily detected by a malware detector. In addition, a malware detector like MalConv [9] takes only the first 2 MB of the file for detection, making appended bytes after 2 MB ineffectual for detection decision. To solve this problem, Suciu et al. [10] discovered regions in the executable that are not mapped to memory and used these slack regions to insert perturbations, as shown on the right side of Figure 2. These slack regions use existing empty regions inside a file placed due to a mismatch between the raw and virtual size of the file. The architecture requires an amount of space used in memory to be multiple of the page size of the operating system. So, it's very likely that some empty space is available on the memory's last page, which has been exploited to store malicious code. However, there is no guarantee that enough space is available to insert perturbation. And some files may not have any slack regions within them.

Demetrio et al. [4] claimed that machine learning models are learning meaningful features from header regions of a PE file rather than the body of a file. So they could generate adversarial binaries by changing only a few tens of bytes of file header as shown in the left of Figure 3. This work did well in terms of eficiency, as few perturbations were enough to overturn the decision of the malware detector. However, with increased eficiency, this approach also increased the chances of breaking the file while manipulating header contents. Conversely, the perturbations injected in header regions are more likely to get detected by a malware detector.

In another work, Yuste et al. [11] introduced the idea of injecting code caves inside malware binaries as in the right of Figure 3. The code caves are not mapped to memory allowing free manipulation of bytes without affecting the file. This allows for increasing the

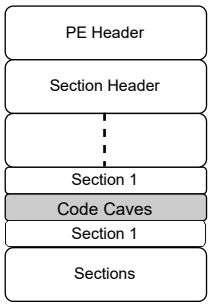

Figure 4: Code Caves within a Section

size of a given binary dynamically. The approach inserts code caves between the PE file's two sections. Since the approach works on unused space and extends the existing space, it gives flexibility over previously discussed approaches. However, there are still a few limitations to this work. Malware detectors can easily detect the perturbations between the sections as they become an obvious place to hide malicious content. In addition, adding the spaces also alters the relative virtual addresses of contents and may lead to breaking down the file.

4 PROPOSED APPROACH

To address all the challenges discussed above, we try to exploit a Windows PE Structure differently. We propose a technique to insert code caves within the sections of a file. Inserting code cave within a section gives a major advantage of hiding the perturbations inside a file's original content. Modifying the contents inside the section is full of challenges, as the chances of breaking the file are very high. To resolve this, we introduce a metamorphic behavior to a malware file, where the file restores its originality on execution. Once the code cave is created in a dynamic environment, any adversarial samples can be optimized inside it without worrying about preserving the file. The steps involved in our approach are as follows:

- Reverse engineer a malware file for its detailed analysis on available slack space, sections present, size of the section, section characteristics
- Record the structural details of a file and choose the target section for code caves
- Dynamically insert the required code cave of size available inside the given section
- Transfer the original content of the cave to other locations, including the end of the file or in slack regions present anywhere inside the file
- Modify the characteristics flag for permissions of different sections as required
- Insert shell code that will restore the original program, i.e., overwrite the code cave with the original contents of the file
- Shell-code should pass the control of a program to the original entry point at the end
- Optimize the code caves and insert more caves if required till evasion or threshold condition

The order of steps above might shufle in specific cases. Since the file is restored to its original form at the end, we don't need to worry about breaking it. Code caves and metamorphic behavior allow us to insert anything inside the caves without altering the program behavior. This shellcode generation for restoring the file is a big challenge in this approach. The shellcode can differ based on the CPU architecture and changing relative virtual addresses. There could be different ways to resolve this, but we are disabling the base relocation of a file. Regardless of the challenges to inserting code caves and preserving the file, this technique allows dynamic code cave insertion while giving flexibility in the size and location of a cave. Our initial experiments have shown some exciting possibilities, which we will publish in our extension of this work.

5 CONCLUSION

This paper discussed how adversarial evasion attacks could be carried out on Windows PE malware files. We presented all the existing techniques used to make space in a PE file to inject perturbation and also analyzed the shortcomings of different approaches. While discussing adversarial, the detailed description of malware detectors, machine learning models, threat modeling, and adversarial generation algorithms are the most but are out of scope for this work. Describing the existing works and vulnerabilities, we propose a noble approach to introduce adversarial examples inside a Windows PE file by inserting code caves. The approach is currently under analysis and shows some good upshots. This work also re-veals vulnerabilities in Windows PE file structure, which can easily be exploited to carry out adversarial evasion attacks.

ACKNOWLEDGEMENTS

This work is partially supported by the National Science Foundation grants 2025682 at Tennessee Tech University, and 2150297 at North Carolina A&T State University.

REFERENCES

[1] Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. 2021. A survey on adversarial attacks for malware analysis. arXiv preprint arXiv:2111.08223 (2021).
[2] Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. 2022. Analysis of Label-Flip Poisoning Attack on Machine Learning Based Malware Detector. In 2022 IEEE International Conference on Big Data (Big Data). IEEE, 4236–4245.
[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13. Springer, 387–402.
[4] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2019. Explaining vulnerabilities of deep learning to adversarial malware binaries. arXiv preprint arXiv:1901.03583 (2019).
[5] Ian JGoodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014).
[6] Jeffrey C Kimmel, Andrew D Mcdole, Mahmoud Abdelsalam, Maanak Gupta, and Ravi Sandhu. 2021. Recurrent neural networks based online behavioural malware detection techniques for cloud infrastructure. IEEE Access 9 (2021), 68066–68080.
[7] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In 2018 26th European signal processing conference (EUSIPCO). IEEE, 533–537.
[8] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. 2018. Adversarial examples on discrete sequences for beating whole-binary malware detection. arXiv preprint arXiv:1802.04528 (2018), 490–510.
[9] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. 2018. Malware detection by eating a whole exe. In Workshops at the thirty-second AAAI conference on artificial intelligence.
[10] Octavian Suciu, Scott E Coull, and Jeffrey Johns. 2019. Exploring adversarial examples in malware detection. In 2019 IEEE Security and Privacy Workshops (SPW). IEEE, 8–14.
[11] Javier Yuste, Eduardo G Pardo, and Juan Tapiador. 2022. Optimization of code caves in malware binaries to evade machine learning detectors. Computers & Security 116 (2022), 102643.