BenchCouncil Transactions on Benchmarks, Standards and Evaluations 3 (2023) 100106

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/

Full length article

ERMDS: A obfuscation dataset for evaluating robustness of learning-based malware detection system

Lichen Jia [a,b], Yang Yang [c], Bowen Tang [a,b], Zihan Jiang [d,*]

[a] State Key Lab of Processors, Institute of Computing Technology, CAS, China
[b] University of the Chinese Academy of Sciences, China
[c] Institute of Technology, China University of Petroleum (Beijing), Karamay Campus, China
[d] Huawei Beijing Research, China

ARTICLE INFO

Keywords:
Dataset
Malware detection system
Security
Machine learning
Adversarial malware

ABSTRACT

Learning-based malware detection systems (LB-MDS) play a crucial role in defending computer systems from malicious attacks. Nevertheless, these systems can be vulnerable to various attacks, which can have significant consequences. Software obfuscation techniques can be used to modify the features of malware, thereby avoiding its classification as malicious by LB-MDS. However, existing portable executable (PE) malware datasets primarily use a single obfuscation technique, which LB-MDS has already learned, leading to a loss of their robustness evaluation ability. Therefore, creating a dataset with diverse features that were not observed during LB-MDS training has become the main challenge in evaluating the robustness of LB-MDS.

We propose a obfuscation dataset ERMDS that solves the problem of evaluating the robustness of LB-MDS by generating malwares with diverse features. When designing this dataset, we created three types of obfuscation spaces, corresponding to binary obfuscation, source code obfuscation, and packing obfuscation. Each obfuscation space has multiple obfuscation techniques, each with different parameters. The obfuscation techniques in these three obfuscation spaces can be used in combination and can be reused. This enables us to theoretically obtain an infinite number of obfuscation combinations, thereby creating malwares with a diverse range of features that have not been captured by LB-MDS.

To assess the effectiveness of the ERMDS obfuscation dataset, we create an instance of the obfuscation dataset called ERMDS-X. By utilizing this dataset, we conducted an evaluation of the robustness of two LB-MDS models, namely MalConv and EMBER, as well as six commercial antivirus software products, which are anonymized as AV1-AV6. The results of our experiments showed that ERMDS-X effectively reveals the limitations in the robustness of existing LB-MDS models, leading to an average accuracy reduction of 20% in LB-MDS and 32% in commercial antivirus software. We conducted a comprehensive analysis of the factors that contributed to the observed accuracy decline in both LB-MDS and commercial antivirus software. We have released the ERMDS-X dataset as an open-source resource, available on GitHub at https://github.com/lcjia94/ERMDS.

1. Introduction

With the rapid progression of technology, machine learning is becoming increasingly sophisticated, prompting researchers to investigate its potential in detecting malware [1–5]. In recent years, machine learning techniques have been employed by researchers to devise more effective approaches for malware detection. One of the key advantages of utilizing machine learning for this purpose is its ability to attain high accuracy rates. This is attributed to the capacity of machine learning models to recognize patterns in malware code that may not be evident to human experts. By training these models on extensive datasets of known malware, researchers can formulate algorithms capable of identifying new malware variants that have not been encountered previously.

However, as the number and complexity of malware threats escalate, conventional signature-based detection methods are losing efficacy. To confront this challenge, commercial antivirus software providers are increasingly embracing machine learning techniques for malware detection [6,7]. By integrating machine learning algorithms into their software, these vendors can elevate the accuracy of their detection capabilities and keep pace with new and emerging threats.

Despite the promising outcomes of using machine learning for malware detection, it is crucial to acknowledge that these models are

* Corresponding author.
E-mail address: jiangzihan.ict@huawei.com (Z. Jiang).

https://doi.org/10.1016/j.tbench.2023.100106
Received 28 March 2023; Received in revised form 23 April 2023; Accepted 23 April 2023
Available online 5 May 2023

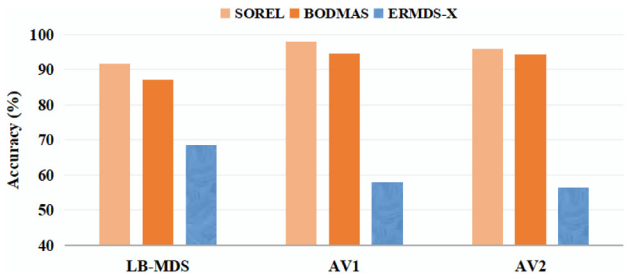


Fig. 1. Accuracy of a LB-MDS (EMBER) and two commercial antivirus softwares (AV1 and AV2) on ERMDS-X (Our dataset) and SOTA malware PE datasets (SOREL and BODMAS).

not impervious to adversarial attacks. Adversarial attacks are methods employed to deceive machine learning models by introducing subtle modifications to input data, with the intention of causing the model to misclassify the data.

Software obfuscation techniques can aid malwares in circumventing detection by LB-MDS. By utilizing obfuscation methods such as encryption and code virtualization, software obfuscation can alter malware features, including file size and API calls, thereby shielding it from detection by LB-MDS.

The concept of robustness in the field of malware detection refers to the ability of detection systems to identify various types of malware and withstand different adversarial attacks. However, the existing PE malware datasets are not sufficient to evaluate the robustness of LB-MDS. To address this, we conducted experiments using two state-of-the-art (SOTA) PE malware datasets, SOREL-20M and BODMAS, on an LB-MDS and two commercial antivirus software, AV1 and AV2. Commercial antivirus software can be categorized as LB-MDS as well, but they do not solely depend on machine learning to identify whether a program is malicious. To differentiate them from the machine learning-based malware detection systems, the term LB-MDS specifically refers to systems that rely entirely on machine learning. Hence, commercial antivirus software is still known as commercial antivirus software. The results are shown in Fig. 1. As seen in Fig. 1, LB-MDS and the two commercial antivirus softwares achieved an average accuracy of over 90% on both the SOREL and BODMAS datasets. Due to the fact that the malware in these datasets employs only a singular obfuscation technique, the features introduced by such obfuscation have already been learned by LB-MDS [8,9]. Consequently, it is impossible for these datasets to diminish the accuracy of LB-MDS. Therefore, they cannot be used to evaluate the robustness of LB-MDS. Since the features in these datasets have been fully assimilated by LB-MDS, rendering them ineffective in reducing the accuracy of the model and thus unsuitable for assessing its robustness.

Numerous researchers have proposed techniques for attacking LB-MDS [10–19]. MAB [10] suggested a series of actions, such as adding a new section to a PE file, and employed reinforcement learning algorithms to select a set of actions that could be applied to malware to produce adversarial examples. Similarly, MalFox [11] introduced a technique in which malware is encrypted and stored in a benign program's section, which is subsequently decrypted and executed at runtime. These approaches reveal the susceptibility of LB-MDS to adversarial attacks.

However, there are several issues associated with the use of these techniques to assess the robustness of LB-MDS. Firstly, the sample size employed in these methods is often insufficient to provide an equitable evaluation of the system's robustness. Typically, 100-1000 malware samples are utilized as input to generate corresponding adversarial examples, which may not accurately represent the extensive range of malware variants that exist in the real world. Secondly, these methods frequently rely on specific techniques, such as the use of encryption in the case of MalFox, to assess the system's robustness, which may not precisely reflect the system's ability to detect other types of obfuscation techniques, such as instruction substitution. Given that commercial antivirus software plays a critical role in security, it is essential to establish a standardized dataset and evaluation methodology to appraise the robustness of LB-MDS.

To address the robustness evaluation problem of LB-MDS, we propose the ERMDS dataset. Unlike prior methods, ERMDS aims to provide a more realistic evaluation of model performance by including a wide array of model-agnostic adversarial examples. These examples are designed to capture various failure modes of modern models, instead of exclusively focusing on worst-case scenarios. When designing this dataset, we created three types of obfuscation spaces, corresponding to binary-level obfuscation, source code-level obfuscation, and packing obfuscation. Each obfuscation space has multiple obfuscation techniques, each with different parameters. The obfuscation techniques in these three obfuscation spaces can be used in combination and can be reused. This enables us to theoretically obtain an infinite number of obfuscation combinations, thereby creating malwares with a diverse range of features that have not been captured by LB-MDS.

To evaluate the ability of the ERMDS obfuscation dataset, we used the obfuscation spaces to generate an instance of the obfuscation dataset called ERMDS-X. This dataset comprises 86,685 malware samples and 30,455 benign samples, with each sample labeled as either malicious or benign. We then used this dataset to evaluate two SOTA LB-MDS models (malConv [2] and EMBER [1]) and six commercial antivirus softwares (AV1-AV6). Through experimentation, we found that ERMDS-X can reduce the accuracy of LB-MDS by an average of 20%, and reduce the accuracy of commercial antivirus software by an average of 32%. By subjecting these detectors to a diverse set of samples, we were able to evaluate their resilience to different types of adversarial attacks and identify areas for improvement. The findings and insights gained from this evaluation are summarized in Table 1, which can inform the design of future LB-MDS.

Apart from presenting the ERMDS dataset, we have also discussed ways to enhance the robustness of malware detectors. We believe that the ERMDS dataset and the proposed methods to improve the robustness of malware detectors will be valuable resources for future research in developing more effective and resilient MDS. By enhancing the robustness of these systems, we can better protect users and organizations from the constantly evolving threat of malware and other cyber attacks.

In summary, this paper has made the following contributions:

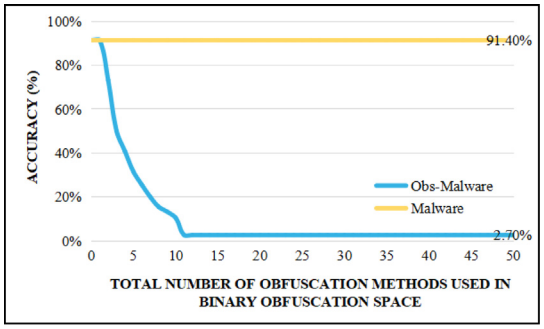
- We propose the ERMDS obfuscation dataset to address the problem that the existing PE malware dataset cannot be used to evaluate the robustness of LB-MDS. We provide a reference implementation of the dataset, ERMDS-X.
- The ERMDS dataset can be utilized for evaluating robustness, and in this study, we utilized ERMDS-X to evaluate the robustness of two LB-MDS models, namely MalConv and EMBER, as well as six commercial antivirus software products, which are anonymized as AV1-AV6. Our experimental results indicate that current LB-MDS models are susceptible to adversarial examples, underscoring the need to enhance their robustness. We have summarized the observations of the LB-MDS systems on the ERMDS dataset in Table 1 and analyzed the underlying reasons for the eight observations.
- We have discussed strategies for improving the robustness of current LB-MDS. We have released the ERMDS-X dataset as an open-source resource.

2. Background

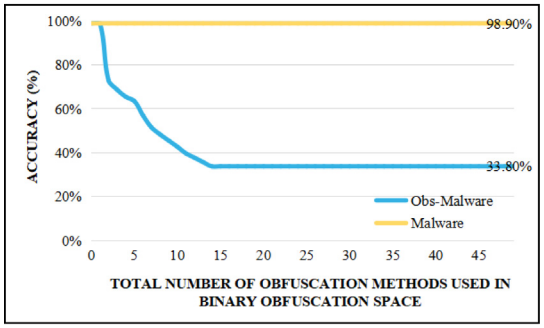
Numerous datasets containing PE malware have been utilized in malware detection research. The EMBER dataset [1], which was introduced in 2018, was the first standardized dataset created specifically for this purpose. It includes 80,000 malware samples collected

Table 1
A summary of major observations and insights grouped by section of the paper.

Observation	Proof	Insight/Explanation
The performance of LB-MDS and commercial antivirus software on the ERMDS-X dataset is much worse compared to their performance on the Clean dataset.	Table 5	Software obfuscation techniques can affect the features of both malicious and benign programs, leading to incorrect conclusions by LB-MDS and commercial antivirus software.
Binary-level obfuscation can significantly reduce the accuracy of LB-MDS by 60%–90%.	Fig. 2	Machine learning models are vulnerable to adversarial attacks, and binary-level obfuscations can more easily create effective adversarial examples.
Binary-level obfuscation only results in a 30% decrease in accuracy for commercial antivirus software.	Fig. 3	Binary obfuscation techniques only have limited ability to modify the code and data of the original program.
LB-MDS will mistake benign programs as malwares.	Fig. 4	Training on more benign program features can reduce false positives.
Source code-level obfuscation increases the probability of misjudging benign programs by LB-MDS.	Fig. 4	Source code-level obfuscation makes benign program code control flow more complex, which leads to misjudgment by LB-MDS.
The misjudgment rate of commercial antivirus software for benign programs is low.	Fig. 4	When unsure, commercial antivirus software tends to classify a program as benign.
Packing technology only decreases the accuracy of EMBER by about 10%.	Fig. 5	Packed programs can be identified by LB-MDS as containing unpacking code, which is considered a feature of malicious programs. This leads to a higher false positive rate for benign programs.
Packing technology can decrease the accuracy of commercial antivirus software by about 60%.	Fig. 6	Packing can completely conceal the features of malicious programs, and benign programs also use packing technology to protect privacy, making it difficult for commercial antivirus software to determine whether a program is malicious based on the presence or absence of unpacking code.



(a) MalConv



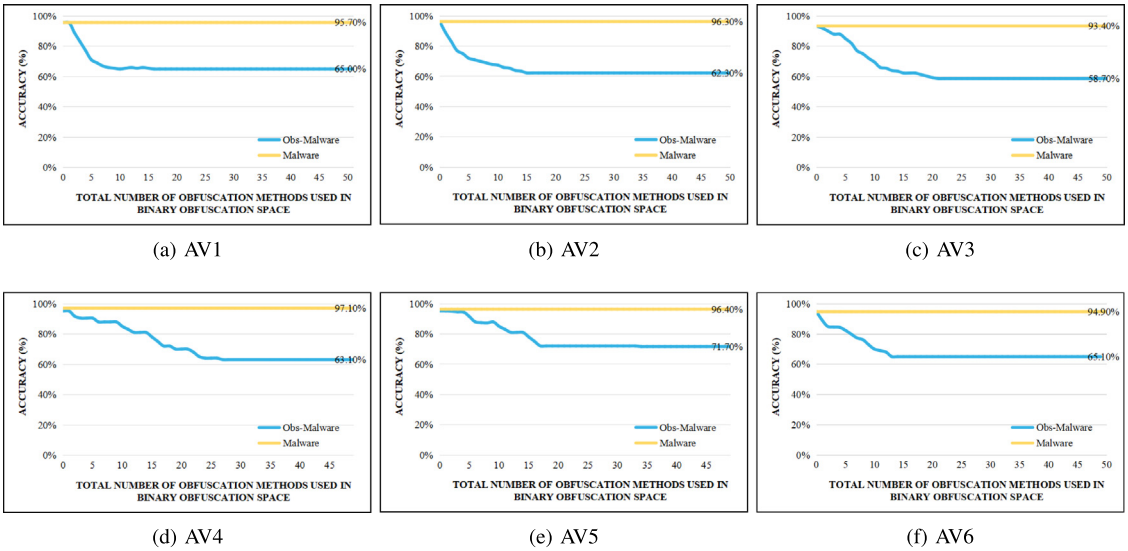
(b) EMBER

Fig. 2. The accuracy of LB-MDS on samples processed through binary obfuscation space.

Algorithm 1: Dataset instance generation algorithm.

Input: Malware dataset $malSet$, Num_b, Num_s, Num_p, l_b, r_b, l_s, r_s, l_p, r_p
Output: Obfuscation dataset $obSet$
$obSet \leftarrow \{\}$
for *each malware sample* $mal \in malSet$ **do**
  for $i = 1$ *to* N_b **do**
    $k \leftarrow randInt(l_b, r_b)$;
    $mal_{binary} \leftarrow$ Applying k rounds of binary obfuscation techniques from Table 2 to the malware;
    $obSet.append(mal_b)$
  end
  for $i = 1$ *to* N_s **do**
    $k \leftarrow randInt(l_s, r_s)$;
    $mal_{source} \leftarrow$ Applying k rounds of source code obfuscation techniques from Table 2 to the malware;
    $obSet.append(mal_s)$
  end
  for $i = 1$ *to* N_p **do**
    $k \leftarrow randInt(l_p, r_p)$;
    $mal_{pack} \leftarrow$ Applying k rounds of packing obfuscation techniques from Table 2 to the malware;
    $obSet.append(mal_{pack})$
  end
end
return $obSet$;

from 2017 to 2018, in addition to 750,00 benign files. Similarly, the SOREL-20M [20] dataset, released in 2019, contains 9 million malware samples collected from 2017 to 2019, as well as 9 million benign files. Although both datasets classify their samples as either malicious or benign, the malware samples in these datasets are sourced from detection websites such as VirusTotal. The malicious samples in these PE malware datasets are relatively outdated and may not represent the latest features of malicious samples. Additionally, the sample features in these datasets can be recognized by LB-MDS and therefore cannot be used to evaluate the robustness of LB-MDS.



(a) AV1 (b) AV2 (c) AV3

(d) AV4 (e) AV5 (f) AV6

Fig. 3. The accuracy of AVs on samples processed through binary obfuscation space.

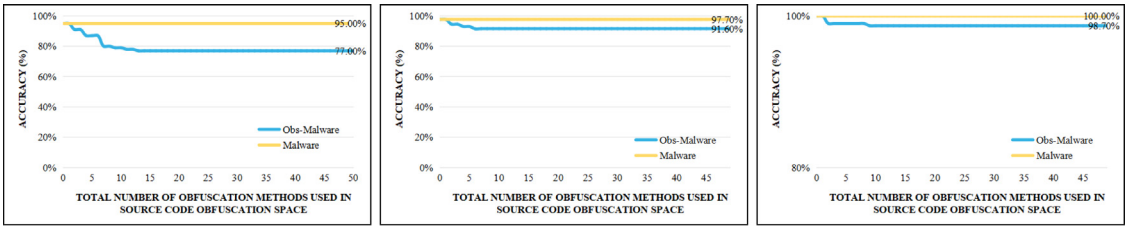
Table 2
Obfuscation methods.

Category	Name	Abbr	Description
Binary Level Obfuscation	Overlay Append	OA	Add additional sections at the end of a binary.
	Section Append	SP	Add randomly generated data to the unused space at the end of a section.
	Section Add	SA	Inserting new sections within the header of a binary.
	Section Rename	SR	Change the name of a section.
	Remove Certificate	RC	Remove the signed certificate.
	Remove Debug	RD	Remove the debug information.
	Break Checksum	BC	Zero out the checksum value.
	Code Randomization	CR	Replace instructions with semantically equivalent instructions.
Source Code Level Obfuscation	Instruction Substitution	IS	This technique involves replacing standard instructions with equivalent but less recognizable ones.
	Code Reordering	CR	This technique involves reordering the instructions in the code to make it harder for attackers to understand the logic of the code.
	Code Flattening	CF	This technique involves converting multi-level if-else statements into a single-level structure.
	Data encryption	DE	This technique involves encrypting sensitive data in the code, such as passwords, keys, and configuration files.
	Code obfuscation through comments	COTC	This technique involves adding comments to the code that are misleading or irrelevant, or that contain obfuscated information.
	Code Metamorphism	CM	This technique involves dynamically modifying the code at runtime, such as by generating code on-the-fly or by modifying the code in memory.
	Control Flow Flattening	CFF	This technique involves modifying the control flow of a program by introducing multiple conditional branches that can be executed in a random order.
	Variable Merging	VM	Combining multiple variables into a single variable to make the code harder to understand.
	Variable Splitting	VS	Splitting a variable into multiple variables to make the code harder to understand.
	Symbol Renaming	SR	Renaming variables, functions, and classes to random or meaningless names to make it harder for a human to understand their purpose and relationships.
	Junk Code Insertion	JOI	Inserting useless or redundant code into the application, making it harder to understand the function of the code.

(*continued on next page*)

Table 2 (*continued*).

Packing	Code Encryption	CE	This technique involves encrypting the executable code of a program in order to prevent it from being understood or modified by an unauthorized user.
	Code Virtualization	CV	This technique involves translating code into specific intermediate representations instead of native instructions and interpreting these representations during runtime.
	Binary Packing	BP	This technique will pack the program. During runtime, the packed program will be unpacked with a custom loader.
	Binary Packing to Benign	BPB	This technique will pack the program and store the packed program in a section of the benign program. During runtime, the packed program will be unpacked with a custom loader.
	API Obfuscation	AO	Hiding the function names and features used by an application programming interface (API), making it harder to understand how the code works.
	Code Compression	CC	Removing unnecessary characters such as whitespace, comments, and newlines to reduce the size of the code and increase analysis difficulty.
	Dynamic Loading	DL	Dividing the code into multiple modules and dynamically loading them when needed to increase code complexity and analysis difficulty.
	Anti-debugging	AD	Adding anti-debugging techniques to the code, such as detecting debuggers or changing the program's execution flow to prevent attackers from debugging and analyzing.
	Anti-decompilation	AP	Adding anti-decompilation techniques to the code, such as adding fake code and control flow to make the results of decompilation unusable.
	Anti-tampering	AT	These are techniques used to detect and prevent modifications to the code. Examples include checking the checksum or hash of the code.
	Anti-disassembly techniques	AS	These are techniques used to prevent an attacker from disassembling the code.
	Anti-emulation	AE	These are techniques used to prevent an attacker from running the code in an emulator.
	Self-modifying code	SMC	This technique involves modifying the code at runtime, making it more difficult to analyze or modify the code.
	Anti-memory Dumping	AMD	These are techniques used to prevent an attacker from dumping the contents of memory to analyze the code. Examples include encrypting memory.



(a) MalConv (b) EMBER (c) AV1-AV6

Fig. 4. The accuracy of LB-MDS on samples processed through source code obfuscation space.

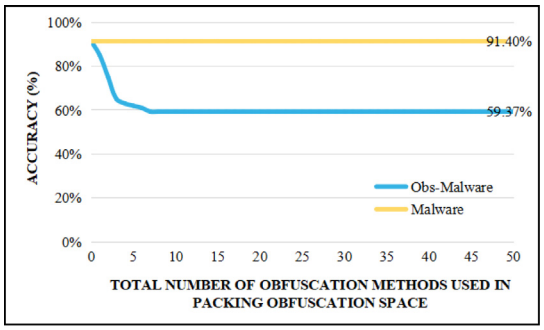
In contrast, the BODMAS dataset [21], released in 2020, contains 70,000 malware samples collected from 2019 to 2020 and 50,000 benign files. Unlike the previous two datasets, the malware samples in BODMAS are labeled with the specific type of malware they belong to, such as ransomware, trojan, or backdoor. This dataset is primarily intended to facilitate LB-MDS in identifying the malware type to which a sample belongs. Hence, it is also not appropriate for evaluating the robustness of LB-MDS.

2.1. Software obfuscation methods

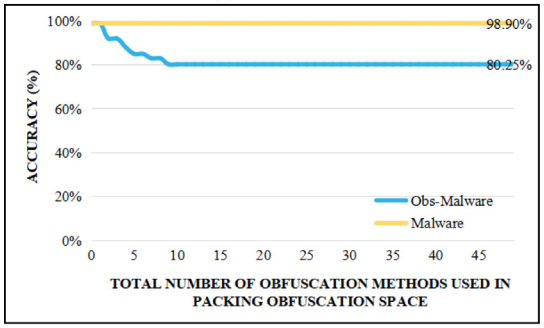
Software obfuscation techniques are utilized to safeguard software code against reverse engineering and analysis, thereby increasing the difficulty for adversaries to comprehend the software's functionality and internal mechanisms. These techniques alter malware while maintaining its functionality, causing LB-MDS to perceive the malicious software as benign programs. Software obfuscation techniques can be categorized into data obfuscation, dynamic code rewriting, and static code rewriting, as surveyed in [22].

2.1.1. Data obfuscation

Data obfuscation techniques [23] involve splitting or merging program data to hinder attackers from analyzing data in the program. For instance, variable splitting splits variables in the program, such as arrays, into multiple sub-arrays. Before accessing the array, the sub-arrays are combined to reform the original array.



(a) MalConv



(b) EMBER

Fig. 5. The accuracy of LB-MDS on samples processed through packing obfuscation space.

2.1.2. Dynamic code rewriting

Dynamic code rewriting techniques [9,24] involve modifying the code at runtime, enabling the dynamic alteration of program behavior. Examples of such techniques include the usage of packers like Ultimate Packer for Executables (UPX) [25] and Themida [26], which apply code obfuscation by encrypting and unpacking the binary program during runtime. Another example is SubVirt [27], which employs code virtualization. This technique transforms the program's code into a specific intermediate representation and interprets this representation at runtime to achieve the same functionality as the original program.

2.1.3. Static code rewriting

Static code rewriting techniques involve transforming the program's code during compilation, eliminating the need for additional modifications during runtime. One such technique is instruction substitution [23,28], which replaces instructions or instruction sequences with semantically equivalent alternatives. For instance, on the Intel x86 platform, the instruction *ADD EAX, 0x1* can be substituted with *SUB EAX, -0x1*.

Dead code insertion [23,28,29] involves constructing code sections that are never executed and injecting code into those sections. This technique aims to confuse or mislead reverse engineers by introducing code that serves no functional purpose.

Control flow obfuscation [28] techniques modify the program's control flow. Control flow flattening, for example, rearranges the program's basic blocks using a switch-case statement. This makes the control flow less transparent and harder to comprehend.

3. The ERMDS dataset

3.1. Methodology

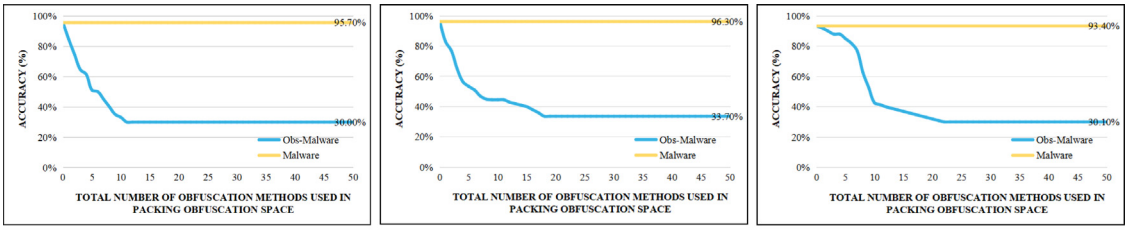
When designing ERMDS, we focused primarily on three questions:

- (q1) How to generate a dataset with diverse features that were not observed during LB-MDS training?
- (q2) How to ensure that each malicious sample has multiple adversarial samples with different features?
- (q3) How to ensure that the functionality of adversarial samples is consistent with that of the original samples?

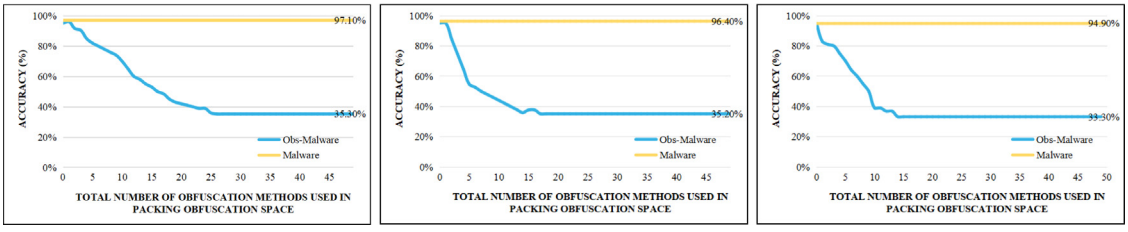
The first question was raised because if all the features in the dataset have already been learned by LB-MDS, then all the samples in the dataset will be correctly classified by LB-MDS, making it impossible to evaluate the robustness of LB-MDS and to determine which LB-MDS is more suitable for security-related applications.

The second question was raised to ensure that every sample in the dataset has multiple adversarial examples with diverse features, as ERMDS aims to provide a more realistic evaluation of model performance by including a wide array of model-agnostic adversarial examples. These examples are designed to capture various failure modes of modern models, instead of exclusively focusing on worst-case scenarios.

The third question was raised because we need to ensure that the samples in the ERMDS dataset are normal executable programs with intact functionality, so that the decrease in LB-MDS accuracy is not due to damaged sample functionality.



(a) AV1 (b) AV2 (c) AV3



(d) AV4 (e) AV5 (f) AV6

Fig. 6. The accuracy of AVs on samples processed through binary obfuscation space.

Table 3
Affected features by Obfuscation methods.

		Hash-Based features		Rule-based features							Data distribution
		File hash	Section hash	Section count	Section name	Section padding	Debug info	Checksum	API calls	Code sequence	Data distribution
Binary Level Obfuscation	OA	✓									✓
	SP	✓	✓			✓					
	SA	✓	✓	✓	✓						✓
	SR	✓			✓						
	RC	✓									
	RD	✓	✓	✓			✓				
	BC	✓						✓			
	CR	✓	✓							✓	
Source Code Level Obfuscation	IS	✓	✓							✓	
	CR	✓	✓							✓	
	CF	✓	✓	✓						✓	
	DE	✓	✓								✓
	COTC	✓	✓				✓				
	CM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	CFF	✓	✓	✓	✓				✓	✓	✓
	VM	✓	✓			✓				✓	✓
	VS	✓	✓			✓				✓	✓
	SR	✓			✓						
	JOI	✓	✓	✓	✓	✓			✓	✓	✓
Packing Obfuscation	CE	✓	✓	✓	✓	✓			✓	✓	
	CV	✓	✓	✓	✓	✓			✓	✓	✓
	BP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	BPB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	AO	✓	✓						✓	✓	✓
	CC	✓	✓	✓	✓	✓			✓	✓	
	DL	✓	✓	✓					✓	✓	✓
	AD	✓	✓						✓	✓	✓
	AP	✓	✓						✓	✓	✓
	AT	✓	✓						✓	✓	✓
	AS	✓	✓						✓	✓	✓
	AE	✓	✓						✓	✓	✓
	SMC	✓	✓	✓	✓	✓		✓	✓	✓	✓
	AMD	✓	✓	✓	✓					✓	✓

3.2. The obfuscation space of ERMDS

Solution for q1. To address the first question, we designed three layers of obfuscation, which correspond to binary-level obfuscation, source code-level obfuscation, and packing obfuscation. Binary-level obfuscation modifies section names and removes debug tables from the malware to eliminate sensitive information. This makes it difficult for the MDS to identify the malware as harmful by matching specific strings or symbol information. Source code-level obfuscation rewrites the control or data flow of the malware to avoid detection based on its code or data features. This method is more advanced than binary-level obfuscation because it rewrites the code and data. Packing is the most advanced method that encrypts the entire malware, incorporates it into a benign program, and decrypts and executes it during runtime. Compared to the other two methods, packing is the most sophisticated technique since the MDS cannot obtain any features of the malware through static detection since it is encrypted.

As shown in Table 2, each layer of obfuscation includes at least eight different obfuscation methods, each with specific parameter settings. These obfuscation methods are orthogonal and can be applied repeatedly, allowing for theoretically infinite combinations of obfuscation methods. This enables us to generate a dataset with diverse features that were not observed during LB-MDS training.

In order to ensure that each obfuscation method in our obfuscation spaces can cover all features of the malware, we first divided the features of the program into Hash-based features, Rule-based features, and Data Distribution according to [10]. We also annotated which features each method would affect. If an obfuscation method *om* modifies the feature set $S = \{s_1, s_2, \ldots, s_k\}$ of a malware sample, the impact of various obfuscation methods on the affected features can be observed in Table 3 for our dataset. For example, consider the CR obfuscation method, which replaces instruction sequences with semantically equivalent ones. Referring to Table 3, we can see that this method affects the File Hash, Section Hash, and Code Sequence features. File Hash and Section Hash are affected by any modifications made to the file and section content, while Code Sequence is only affected if the CR obfuscation method modifies the instruction sequences in the code. As shown in Table 3, each obfuscation method in our obfuscation spaces can cover all features of the program, ensuring that ERMDS can generate malware with diverse features.

Solution for q2. For problem two, since the obfuscation combinations we generate are theoretically infinite, each combination applied to a malware produces a variant of the original malware. Therefore, ERMDS can theoretically produce an infinite number of variants for each malicious sample, ensuring that each sample has multiple adversarial samples with different features.

Solution for q3. Regarding problem three, the obfuscation methods selected in our obfuscation space are functionality-preserving. In theory, applying these methods to a program should not change its original functionality because these methods, as shown in Table 2, are all designed to preserve the program's functionality [10,11]. However, due to implementation issues, a small number of programs may become non-functional. To quantify the impact of implementation errors on program functionality, we conducted additional experiments to evaluate the effect of obfuscation methods in the ERMDS dataset on program functionality. Through this experiment, we discovered that the obfuscation methods in the ERMDS dataset can indeed affect the original functionality of functions. For details, please refer to the Functional Integrity Testing section in the appendix. Furthermore, other researchers can expand the ERMDS by incorporating additional obfuscation techniques, resulting in a more comprehensive and diverse dataset of malware samples.

Table 4
Overview of ERMDS-X dataset and quality thresholds on four datasets.

Level	malware nums	benign nums	Quality threshold of LB-MDS accuracy %	Quality threshold of commercial antivirus softwares accuracy %
Binary obfuscation	49714	16815	18.25	64.32
Source Code obfuscation	0	3841	84.3	98.7
Packing	36971	9799	79.81	32.93
ERMDS-X	86685	30455	62.35	62.51

3.3. Workflow of ERMDS

Algorithm 1 outlines the workflow of the ERMDS, which generates a obfuscation dataset $obSet$. For each malware sample mal in the $malSet$, three types of obfuscation techniques, namely binary obfuscation, source code obfuscation, and packer, are applied to mal to produce a set of obfuscated malware samples, which are then added to $obSet$. Num_b, Num_s, and Num_p denote the number of binary obfuscation samples, source code obfuscation samples, and packer samples that need to be generated for each mal, respectively. When generating the obfuscation samples, we randomly choose which obfuscation techniques to apply. To ensure the diversity of the generated samples, we perform k rounds of selection for each obfuscation technique, resulting in a sequence of obfuscation methods $O = \{O_1, O_2, \ldots, O_k\}$, where each element represents a specific obfuscation method. The value of k is a random number, and l_b and r_b are the upper and lower bounds for binary obfuscation techniques, l_s and r_s are the upper and lower bounds for source code obfuscation techniques, and l_p and r_p are the upper and lower bounds for packer techniques.

4. Implementation

4.1. Initial dataset description

As the majority of current datasets for malware analysis only contain samples from the period between 2017 and 2020, including the most recently released BODMS, there is a risk that these datasets may not accurately reflect recent malware behaviors. To address this issue, we intend to release a new malware dataset that covers samples from January to December 2022. Our initial dataset contains 10,000 malware samples, 5000 benign samples, and 300 benign samples with source codes, totaling 15,300 samples. We collected the malware samples from VirusShare [30], ensuring that they were collected between January 1, 2022, and December 30, 2022. The benign samples were collected from Github and Source Forge.

4.2. Generate ERMDS-X dataset instance

We used Algorithm 1 to generate an instance of our dataset, named ERMDS-X, with the following parameters: $Num_b = 30$, $Num_s = 30$, $Num_p = 30$, $l_b = 2$, r_{50}, $l_s = 2$, $r_s = 50$, $l_p = 2$, and $r_p = 50$. The malware dataset $malSet$ consisted of 4000 malware and 1500 benign samples, which were random sampled from the initial dataset. The parameters used in our ERMDS-X dataset instance, including $Num_b = 30$, $Num_s = 30$, $Num_p = 30$, $l_b = 2$, r_{50}, $l_s = 2$, $r_s = 50$, $l_p = 2$, and $r_p = 50$, were not chosen to minimize the accuracy drop of LB-MDS. Instead, the ERMDS dataset was designed to provide a more realistic evaluation of model performance by incorporating a diverse set of model-agnostic adversarial examples. These examples aim to capture various failure modes of modern models, rather than focusing solely on worst-case scenarios. In the Parameters section of the Appendix, we provide a set of optimal parameters that can minimize the accuracy drop of LB-MDS.

After filtering out some malware that could not be processed, we obtained a total of 86,685 malicious and 30,455 benign samples for the ERMDS-X dataset. We extracted the features from PE files using the LIEF [31] project and followed the same format as Ember [1], SOREL-20M [20], and BODMAS [21] to ensure compatibility with existing datasets. Each sample in the ERMDS-X dataset is labeled either "malware" or "benign", providing a ground-truth label for researchers. The techniques used are listed in Table 2, and were implemented using the following tools: Binary Ninja [32], Radare2 [33], IDA Pro [34], LLVM [35], Pin [36], Angr [37], and MAB [10] for binary obfuscation; OLLVM (Obfuscator-LLVM) [28], PreEmptive Protection - Dotfuscator [38], and ConfuserEx [39] for source code obfuscation; and Themida [26], UPX [25], ASProtect [40], Enigma [41], Virbox Protector [42], VMProtect [43], and MalFox [11] for packing obfuscation.

4.3. Description on ERMDS-X dataset instance

The ERMDS-X dataset serves as a valuable tool for evaluating the robustness of LB-MDS and facilitating the identification of potential vulnerabilities within the LB-MDS system. Moreover, LB-MDS can be trained again on the ERMDS-X dataset to enhance its robustness. An overview of the ERMDS-X dataset is provided in Table 4, which comprises three sub-datasets: the Binary obfuscation dataset, the Packing dataset, and the Source Code obfuscation dataset. Each of these sub-datasets includes samples that have undergone binary-level obfuscation, packing, and source code obfuscation, respectively. The ERMDS-X dataset is the combination of these three sub-datasets.

As a result of the absence of source code in malware, we limited the application of source-level obfuscation to benign programs that have source code. It is possible to achieve better accuracy reduction through source-level obfuscation if there is enough source code available for malware. Although this presents a drawback of ERMDS-X, our experiments have demonstrated that ERMDS-X is sufficient for evaluating the robustness of MDS. Other researchers can readily expand the source-level obfuscation dataset by providing malware with source code.

Additionally, we present the quality threshold of LB-MDS on these four datasets, including the quality threshold of LB-MDS and commercial antivirus software. The quality threshold of LB-MDS is determined by averaging the accuracy of two LB-MDS models, namely MalConv and Ember, selected during our experiments. The quality threshold of commercial antivirus software is determined by averaging the accuracy of six commercial antivirus software programs chosen during our experiments.

5. Evaluation

Our evaluation aims to address the following research questions:

- RQ1: How do LB-MDS and commercial antivirus software perform on the ERMDS-X dataset and the SOTA PE malware datasets?
- RQ2: What is the impact of the three categories of software obfuscation techniques, namely binary-level obfuscation, source code-level obfuscation, and packing, on the effectiveness of LB-MDS and commercial antivirus software?
- RQ3: Which malware features have the greatest impact on the classification results of LB-MDS?

Table 5
Accuracy of learning-based malware detection systems on ERMDS-X and Clean datasets.

Model	$E^{Network}_{Clean}$ (%)	$E^{Network}_{ERMDS}$ (%)
malConv	78.57	56.17
EMBER	85.37	68.53
AV1	94.62	65.18
AV2	96.87	66.29
AV3	95.45	58.01
AV4	95.88	56.45
AV5	93.33	72.83
AV6	96.41	56.31

Table 6
Accuracy of learning-based malware detection systems on SOREL-20M and BODMAS datasets.

Model	$E^{Network}_{SOREL}$ (%)	$E^{Network}_{BODMAS}$ (%)
malConv	87.9	82.2
EMBER	91.7	87.1
AV1	93.7	96.3
AV2	92.5	96.8
AV3	97.9	94.5
AV4	96.1	93.8
AV5	97.3	94.7
AV6	96.0	94.4

5.1. Evaluation metrics

In order to comprehensively evaluate the robustness of a MDS, we begin by selecting an MDS and then calculating its accuracy on a clean dataset, which serves as the initial dataset. This calculation is performed using Eq. (1) according to [44], where E^{MDS}_{Clean} denotes the MDS's accuracy on Clean dataset. Specifically, N^{C}_{Clean} represents the number of samples that are correctly predicted by the MDS, while N^{All}_{Clean} represents the total number of samples in the initial dataset.

$$E^{MDS}_{Clean} = N^{C}_{Clean} / N^{All}_{Clean} \tag{1}$$

Subsequently, the selected MDS is tested on the ERMDS-X dataset (denoted as "ERMDS"), and the accuracy, denoted as E^{MDS}_{ERMDS}, is calculated using Eq. (2) according to [44]. Here, N^{C}_{ERMDS} represents the number of samples that are correctly predicted, while N^{All}_{ERMDS} represents the total number of samples in the ERMDS-X dataset.

$$E^{MDS}_{ERMDS} = N^{C}_{ERMDS} / N^{All}_{ERMDS} \tag{2}$$

5.2. Attack targets

For our target models, we have chosen the following:

- EMBER [1] is an open-source LB-MDS. It utilizes LIEF [31] to extract features from both malicious software and benign programs. These features are then used by a LightGBM model to determine whether a program is malicious or not. We utilized the model provided by MLSEC 2019 as our target for the attack [45].
- MalConv [2], in contrast to EMBER, directly uses the binary byte stream of malicious software as training data. Based on this byte stream, it determines whether a program is malicious. We employed the model provided by MLSEC 2019 as our target for the attack [45].
- Commercial Antivirus Software. We selected six top commercial antivirus software as our evaluation targets, based on [46].

5.3. Evaluation on ERMDS-X dataset

This experiment demonstrates that ERMDS-X effectively exposes the robustness limitations of existing LB-MDS models. In Table 5, a comprehensive comparison of two LB-MDS models and six commercial antivirus software is presented based on their detection performance on both the ERMDS-X and Clean datasets. The Clean dataset is a collection of original data containing both malwares and benign programs. Notably, MalConv exhibits a significantly lower accuracy of only 78.57% on the Clean dataset in comparison to EMBER and the six commercial antivirus software, which all have an accuracy of over 85%. Our analysis suggests that this decrease in accuracy is due to the outdated dataset used by the MalConv model during training, which failed to capture the latest features of malwares. In addition, research conducted in [21] demonstrates that virus features change over time, and previously trained models may have decreased accuracy on new malwares.

The performance of the LB-MDS models and commercial antivirus software on the ERMDS-X dataset demonstrates an accuracy range of 56.17% to 78.83%, with an average accuracy of 62.47%. This lower accuracy is attributed to the different types of adversarial examples present in the ERMDS-X dataset, which aims to provide a more realistic evaluation of model performance by including a broad range of model-agnostic adversarial examples. These adversarial examples are designed to capture various failure modes of modern models, rather than focusing solely on worst-case scenarios. Thus, the performance of the LB-MDS models on the ERMDS-X dataset did not decrease to the lowest level, but an accuracy of 62.47% is still a relatively low value, demonstrating the ability of ERMDS-X to evaluate the robustness of existing LB-MDS models.

It is essential to note that LB-MDS systems predict whether a given software is malicious or benign. Even with random guessing, there is a 50% probability of correctly guessing. The accuracy of these systems on the ERMDS-X dataset is 62.47%, which is only 12.47% higher than random guessing. Thus, ERMDS-X can be used to evaluate the robustness of existing LB-MDS models effectively.

5.4. Evaluation on SOTA malware datasets

This experiment aims to demonstrate that the SOTA malware datasets are not suitable for evaluating the robustness of LB-MDS. We evaluated the accuracy of two SOTA PE malware datasets, SOREL-20M and BODMAS, using two LB-MDS models and six commercial antivirus software. To ensure a fair comparison, we randomly selected 10,000 samples from each dataset and employed them to attack the two LB-MDS models and six commercial antivirus software, with each experiment repeated five times to obtain average results.

Table 6 illustrates the accuracy of the two LB-MDS models and six commercial antivirus software on the SOREL-20M and BODMAS datasets. It is observed that all the commercial antivirus software exhibit an accuracy of over 90% for detecting samples from both SOREL and BODMAS datasets. In contrast to Section 5.3, the accuracy of commercial antivirus software for ERMDS-X ranges from 56.31% to 72.83%. Although the LB-MDS models attain an accuracy of over 80% for detecting samples from both SOREL and BODMAS datasets, their accuracy for ERMDS-X is only 56.17% and 68.53%, respectively. This implies that ERMDS-X can assess the robustness of MDS, whereas other datasets like SOREL and BODMAS cannot significantly impact the accuracy of MDS.

Moreover, as evident from Table 6, malConv and EMBER display lower accuracy on the BODMAS dataset than on the SOREL dataset. Additionally, Section 5.3 indicates that malConv and EMBER exhibit lower accuracy on the clean dataset. This is due to the fact that the clean dataset was gathered in 2022, which is subsequent to the data collection period of SOREL and BODMAS datasets (2017–2020). EMBER and malConv were trained on a relatively outdated dataset before 2018 and were unable to accurately capture the features of new viruses, resulting in their lower accuracy.

5.5. Evaluation on three obfuscation spaces

In order to evaluate the effectiveness of LB-MDS on three types of obfuscation spaces, namely binary obfuscation space (BOS), source code obfuscation space (SOS), and packing obfuscation space (POS), we generated adversarial examples using each of these three techniques and evaluated the accuracy of LB-MDS. Furthermore, we conducted an analysis on the samples that caused a decrease in the accuracy of LB-MDS in each obfuscation space, counted the frequency of each obfuscation method used in each obfuscation space, and included a set of parameters in the appendix that can induce the maximum decrease in LB-MDS accuracy, which can serve as a reference for other researchers.

5.5.1. Effect on binary obfuscation space

To evaluate the quality of adversarial examples generated by the BOS, we randomly selected 10000 malicious samples from the initial dataset and named it dataset-V. For each malicious sample in dataset-V, we applied obfuscation techniques from the binary obfuscation space iteratively until an effective adversarial example was produced. We used the obfuscation methods from the BOS to attack MalConv, EMBER, and six commercial antivirus software, and evaluated the number of obfuscation iterations needed to generate an effective adversarial example, as illustrated in Fig. 2. To ensure experimental accuracy, each experiment was repeated five times to obtain the average results.

Attacking LB-MDS. Fig. 2 illustrates the efficacy of BOS attacks on EMBER and MalConv. As depicted in Fig. 2(a), MalConv's original accuracy on malware is 91.40%. However, when detecting malware processed by BOS, the accuracy plummets to 2.70%, resulting in an 88.7% decline in the accuracy. In Fig. 2(b), we observe that EMBER's original accuracy on malware is 98.90%. However, after being processed by BOS, the accuracy drops to 33.8%, resulting in a 65.1% reduction in the accuracy. These results highlight that BOS attacks can easily deceive LB-MDS.

Furthermore, we evaluated the relationship between the number of obfuscation methods and the accuracy of LB-MDS. As depicted in Fig. 2, for MalConv, after undergoing ten binary obfuscation methods, the accuracy of malware decreased to its lowest point, dropping from 91.4% to 2.7%. Even when continuing to apply binary obfuscation techniques to the malware, MalConv's accuracy did not further decrease after ten obfuscation methods. For EMBER, after undergoing thirteen binary obfuscation methods, the accuracy of malware decreased to its lowest point, dropping from 98.9% to 33.8%. Even when continuing to apply binary obfuscation techniques to the malware, EMBER's accuracy did not further decrease after thirteen obfuscation methods. This experiment highlights the limitations of binary obfuscation techniques in combating LB-MDS, as they typically only add new content and have limited ability to modify the code and data of the original program. Therefore, binary obfuscation techniques cannot fully defeat LB-MDS.

Attacking Commercial Antivirus. We conducted a comprehensive evaluation of our framework using six commercially available antivirus engines. Fig. 3 displays the original accuracy of malware for AV1-AV6, which ranged from 93.40% to 97.10%, with AV4 achieving the highest rate at 97.10% and AV3 the lowest rate at 93.40%. After applying BOS processing to the malware and subjecting it to detection by AV1-AV6, we observed a significant reduction in accuracy for the processed malware. For instance, the accuracy for AV1 dropped from an original 95.70% to 65% after BOS processing, leading to 29.3% of the malware evading detection.

Our findings indicate that while binary obfuscation techniques exhibits a certain level of effectiveness against commercial antivirus software, it is not entirely successful in defeating it. This is mainly because commercial antivirus software uses multiple features to determine whether a program is malicious or benign. Binary obfuscation techniques have limited ability to modify the code and data of a program, such as being unable to modify the API calls feature of a program, leading to suboptimal performance in terms of adversarial effectiveness against commercial antivirus software.

Additionally, we analyzed the impact of the number of binary obfuscation techniques on the accuracy of commercial antivirus software. Fig. 3 shows that the number of binary obfuscation techniques had varying degrees of influence on the accuracy of the different antivirus engines. For example, malware detection probability for AV1 decreased to its lowest value of 65% after the 8th binary obfuscation, while for AV4, it decreased to its lowest value of 63.1% after the 27th binary obfuscation. This indicates that different commercial antivirus software exhibits varying degrees of sensitivity to binary obfuscation.

5.5.2. Effect on source code obfuscation space

To evaluate the quality of adversarial examples generated by the SOS, we were unable to use malware samples due to their lack of source code. Therefore, we only provided benign programs to SOS for obfuscation and tested whether the obfuscated benign programs were misclassified as malware by LB-MDS. We randomly selected 300 benign samples from the initial dataset and named it dataset-B. For each benign sample in dataset-B, we applied obfuscation techniques from the binary obfuscation space iteratively until an effective adversarial example was produced. We used the obfuscation methods from the SOS to attack MalConv, EMBER, and six commercial antivirus software, and evaluated the number of obfuscation iterations required to generate an effective adversarial example, as depicted in Fig. 4. To ensure experimental accuracy, each experiment was repeated five times to obtain the average results.

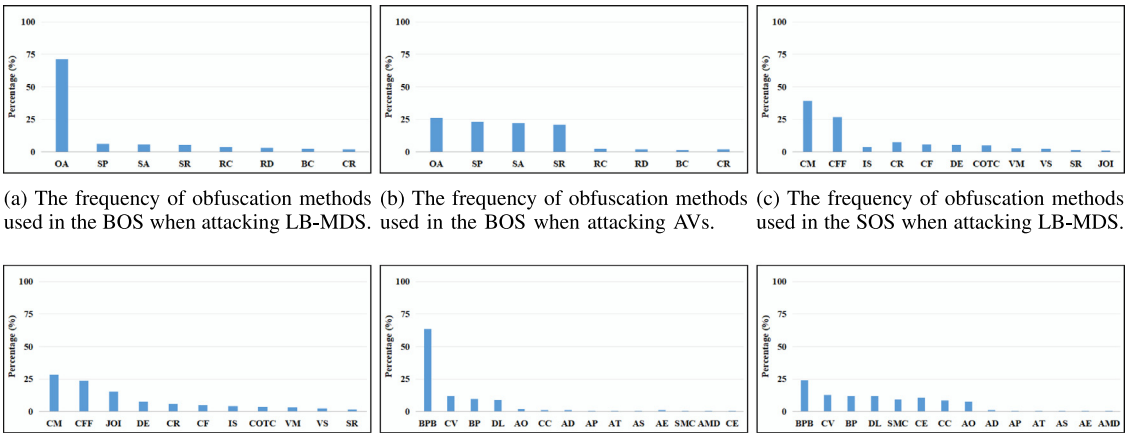
Fig. 5 illustrates the accuracy of EMBER, MalConv, and six commercial antivirus engines in detecting benign programs processed by SOS. MalConv and EMBER exhibit a certain false positive rate for benign programs, with accuracies of 95% and 97.7%, respectively, indicating that MalConv misclassifies 5% of benign programs as malicious and EMBER misclassifies 2.3%, as shown in (a) and (b). After SOS processing, 15 rounds of source code obfuscation cause MalConv's accuracy to decrease to its lowest value of 77%, and 6 rounds of source code obfuscation cause EMBER's accuracy to decrease to its lowest value of 91.6%. This may be due to the fact that source code obfuscation completely disrupts program control flow, making them appear unlike normal programs, leading LB-LDS to misclassify them as malicious.

For commercial antivirus software, the accuracy for benign programs is 100%, and it can also achieve an accuracy of 98.7% for benign programs processed by SOS. This, in conjunction with Table 5, indicates that although LB-LDS has high accuracy for malware, it also has a certain false negative rate for benign programs processed by obfuscation techniques. Furthermore, Fig. 2 shows that LB-LDS is vulnerable to adversarial examples. Therefore, commercial antivirus software outperforms LB-LDS in detecting obfuscations at both the binary and source code levels.

5.5.3. Effect on packing obfuscation space

To ensure fairness, we evaluated the quality of adversarial examples generated by the POS using the same dataset (dataset-V) as in the BOS evaluation. For each malicious sample in dataset-V, we applied obfuscation techniques from the POS iteratively until an effective adversarial example was produced. We used the obfuscation methods from the POS to attack MalConv, EMBER, and six commercial antivirus software. To ensure experimental accuracy, each experiment was repeated five times to obtain the average results.

Attacking LB-MDS. Fig. 5 illustrates the attack effects of the POS on MalConv and EMBER. As shown in the figure, the malicious samples processed by the POS can significantly reduce the accuracy of MalConv from 91.4% to 59.4%. However, the impact on the accuracy of EMBER is relatively small, only reducing its accuracy from 98.9% to 80.25%. The poor performance of the POS on LB-MDS is mainly due to the fact that the POS will pack the malicious software, which requires adding corresponding unpacking code to ensure the correct execution of the malicious software. LB-MDS can capture the features of the



(a) The frequency of obfuscation methods used in the BOS when attacking LB-MDS.
(b) The frequency of obfuscation methods used in the BOS when attacking AVs.
(c) The frequency of obfuscation methods used in the SOS when attacking LB-MDS.
(d) The frequency of obfuscation methods used in the SOS when attacking AVs.
(e) The frequency of obfuscation methods used in the POS when attacking LB-MDS.
(f) The frequency of obfuscation methods used in the POS when attacking AVs.

Fig. 7. The frequency of obfuscation methods used in generating samples that cause a decrease in the accuracy of LB-MDS.

unpacking code, which makes the performance of the POS not ideal when attacking LB-MDS.

Attacking Commercial Antivirus. Fig. 6 illustrates the effectiveness of POS in generating adversarial examples against six commercial antivirus software, namely AV1-Av6. In comparison to LB-MDS, POS demonstrates considerably superior performance in attacking commercial antivirus software. The accuracy of all six commercial antivirus software in detecting original malware samples ranges from 93.4% to 98.9%. However, when presented with malware obfuscated by POS, their accuracy drops dramatically to between 30.1% and 33.7%. This significant decrease in accuracy can be attributed to the fact that many packing obfuscation methods, such as BPB, encrypt the entire malware, resulting in modifications to all features of the malware. Moreover, the encrypted malware is stored in the section of benign programs, causing commercial antivirus software to misclassify it as benign software.

5.5.4. Most frequently used obfuscation methods analysis

In this experiment, we analyzed the samples that caused a decrease in the precision of LB-MDS in the three aforementioned experiments, and calculated the frequency of obfuscation methods used in these samples. Understanding the reasons for the decrease in accuracy of LB-MDS can help improve the robustness of a classifier against adversarial attacks. We have summarized the most frequently used obfuscation methods in Fig. 7 Based on this figure, we can infer the root cause of each evasion. Our findings indicate that:

- When attacking LB-MDS, OA in the binary obfuscation space, CM, CFF in the source code obfuscation space, and BPB in the packing obfuscation space are the most frequently used methods. Other obfuscation methods are rarely used. OA, CM, CFF, and BPB all affect the data distribution of the software, indicating that changes to the data distribution are the main reason for the decrease in LB-MDS accuracy.
- When attacking commercial antivirus software, OA, SP, SA, and SR in the binary obfuscation space are frequently used and mainly affect the program's data distribution, section padding, section name, and section hash features. This indicates that single obfuscation methods are no longer effective in reducing the accuracy of commercial antivirus software, and multiple obfuscation techniques are needed to modify multiple features of the program to decrease the accuracy of commercial antivirus software. Additionally, CM, CFF, JOI, and DE in the source code obfuscation space, and BPB, CV, BP, DL, SMC, CE, CC, and AO in the packing obfuscation space are frequently used, which supports the above conclusion that single obfuscation methods are insufficient.

6. Discussion

In this section, we will discuss methods to enhance the robustness of LB-MDS and analyze the current state of software obfuscation and malware detection. As previously demonstrated, commercial antivirus software exhibits poor performance in detecting packed malware due to encryption, which eliminates the feature and renders feature-based methods ineffective. However, packing can be utilized to protect intellectual property or important data, and simply labeling programs containing unpacking code as malware is not a practical solution. Packed malware necessitates decryption before regular execution, and the decrypted code and data of the malicious program are in plaintext in memory, enabling commercial antivirus software to detect malicious features using feature-based methods. Therefore, we suggest that antivirus software should primarily employ dynamic detection methods when determining whether a program is malicious or benign, as static detection cannot acquire the features of packed programs.

7. Conclusion

This paper presents a obfuscation dataset ERMDS that solves the problem of evaluating the robustness of LB-MDS. To evaluate the ability of the ERMDS obfuscation dataset, we used the obfuscation spaces to generate an instance of the obfuscation dataset called ERMDS-X. We then used this dataset to evaluate two LB-MDS models and six commercial antivirus softwares. Through experimentation, we found that ERMDS-X can reduce the accuracy of LB-MDS by an average of 20%, and reduce the accuracy of commercial antivirus software by an average of 32%. Finally, we analyzed the reasons for the decrease in accuracy for each LB-MDS and commercial antivirus software, and provided suggestions for improving robustness.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix

Parameters. The following parameters constitute a set that maximally reduces the accuracy of LB-MDS: $Num_b = 30$, $Num_s = 30$, $Num_p = 30$, $l_b = 30$, $r_b = 30$, $l_s = 10$, $r_s = 10$, $l_p = 25$, and $r_p = 25$. We recommend that Num_b, Num_s, and Num_p be set to at least 30 to ensure that each sample has multiple adversarial examples. $l_b = 30$ because

Table A.7
Functional testing of obfuscated malicious software.

Malware	*Functional Rate* (%)
malware1	91
malware2	95
malware3	93

over 30 rounds of binary obfuscation can minimize the accuracy of LB-MDS and commercial antivirus software AV1-AV6, as shown in Figs. 2 and 3. $r_b = 30$ because even if the number of binary obfuscation rounds is further increased, the accuracy of LB-MDS and commercial antivirus software will not decrease further after reaching 30 rounds. $l_s = 10$ because over 10 rounds of source code obfuscation can minimize the accuracy of LB-MDS and commercial antivirus software AV1-AV6, as shown in Fig. 4. $r_s = 10$ because even if the number of source code obfuscation rounds is further increased, the accuracy of LB-MDS and commercial antivirus software will not decrease further after reaching 10 rounds. $l_p = 25$ because over 25 rounds of packing obfuscation can minimize the accuracy of LB-MDS and commercial antivirus software AV1-AV6, as shown in Fig. 4. $r_b = 25$ because even if the number of packing obfuscation rounds is further increased, the accuracy of LB-MDS and commercial antivirus software will not decrease further after reaching 25 rounds.

Future Work. Comprehensively evaluating the robustness of MDS is a challenging task. In this paper, we primarily employ three types of obfuscation space to assess the performance of existing MDS under adversarial attacks. The study confirms that obfuscation techniques can be used to evaluate the robustness of MDS. However, the generation of adversarial samples is not limited to obfuscation techniques alone. For instance, in DeepMal [19], adversarial instructions were inserted into malware, allowing the generated adversarial samples to evade detection by CNN-based MDS. Such techniques can effectively capture the vulnerability of LB-MDS since small modifications to malware can deceive LB-MDS. Therefore, in future work, we will incorporate such adversarial attack techniques as an essential approach to generate more diverse samples in the EMBDR dataset and continuously enhance its richness.

Functional Integrity Testing This experiment aims to evaluate whether the combination of obfuscation techniques will compromise the functionality of malicious software. We randomly selected three malicious software programs with clear functionalities: Malware 1 encrypts files on the computer and extorts money, Malware 2 is a Trojan horse program client, and Malware 3 is a malicious advertisement plugin. Since we do not have access to the source code of these malicious software programs, we used obfuscation techniques from BOS and POS that were combined in various ways, with the number of combinations times between 3–15, to process these three malicious software programs. These obfuscation techniques were applied to each malicious software program to generate 100 different obfuscated versions. We then manually executed each obfuscated malicious software program to determine whether their functionalities had been altered. For example, we tested whether the obfuscated extortion software was still capable of encrypting files and extorting money. If the functionalities of these malicious software programs were not altered, it indicated that the combination of obfuscation techniques did not compromise the functionality of the programs.

From Table A.7, we can observe that even after undergoing various obfuscation techniques, most of the malicious software programs were still able to execute their original functionalities. Only 7% of the obfuscated malicious software programs lost their original functionalities after being processed. Analysis of the obfuscated malicious software programs that did not execute correctly led to the following conclusions: 1) To prevent tampering, Malware 1 calculated a checksum of certain parts of their code and checked whether the checksum was correct before execution. If our obfuscation method modified this part of the code, the malicious software would not execute because the checksum would have changed. 2) Malware 2 and 3 had a more complex PE format compared to Malware 1, with a large .rsrc resource section. Existing obfuscation tools such as Malfox could not correctly parse this section when processing Malware 2 and 3, causing it to not execute correctly.

References

[1] H.S. Anderson, P. Roth, EMBER: An open dataset for training static PE malware machine learning models, 2018, ArXiv e-prints arXiv:1804.04637.

[2] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, C.K. Nicholas, Malware detection by eating a whole EXE, 2017, ArXiv arXiv:1710.09435.

[3] G.E. Dahl, J.W. Stokes, L. Deng, D. Yu, Large-scale malware classification using random projections and neural networks, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 3422–3426, http://dx.doi.org/10.1109/ICASSP.2013.6638293.

[4] K. Rieck, P. Trinius, C. Willems, T. Holz_aff2n3, Automatic Analysis of Malware Behavior Using Machine Learning, 19 (4) (2011) 639–668.

[5] J. Saxe, K. Berlin, Deep neural network based malware detection using two dimensional binary program features, in: 2015 10th International Conference on Malicious and Unwanted Software, 2015, pp. 11–20, http://dx.doi.org/10.1109/MALWARE.2015.7413680.

[6] Avast 2018. AI & machine learning, 2018, https://www.avast.com/en-us/technology/aiand-machine-learning.

[7] M.D.A.R. Team., New machine learning model sifts through the good to unearth the bad in evasive malware, 2019, https://www.microsoft.com/security/blog/2019/07/25/new-machine-learning-model-sifts-through-the-good-to-unearth-the-bad-in-evasive-malware/.

[8] S.H. Ding, B.C. Fung, P. Charland, Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization, in: 2019 IEEE Symposium on Security and Privacy, SP, IEEE, 2019, pp. 472–489.

[9] X. Ren, M. Ho, J. Ming, Y. Lei, L. Li, Unleashing the hidden power of compiler optimization on binary code difference: An empirical study, in: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, 2021, pp. 142–157.

[10] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, H. Yin, 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, 2022, pp. 990–1003, http://dx.doi.org/10.1145/3488932.3497768.

[11] F. Zhong, X. Cheng, D. Yu, B. Gong, S. Song, J. Yu, MalFox: Camouflaged adversarial malware example generation based on C-GANs against black-box detectors, 2020, ArXiv, arXiv:2011.01509.

[12] A. Al-Dujaili, A. Huang, E. Hemberg, U.-M. OReilly, Adversarial Deep Learning for Robust Detection of Binary Encoded Malware, 2018, pp. 76–82, http://dx.doi.org/10.1109/SPW.2018.00020.

[13] H. Anderson, A. Kharkar, B. Filar, D. Evans, P. Roth, Learning to evade static PE machine learning malware models via reinforcement learning, 2018.

[14] R.L. Castro, C. Schmitt, G. Dreo, AIMED: Evolving malware with genetic programming to evade detection, in: 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), 2019, pp. 240–247, http://dx.doi.org/10.1109/TrustCom/BigDataSE.2019.00040.

[15] L. Chen, Understanding the efficacy, reliability and resiliency of computer vision techniques for malware detection and future research directions, 2019, ArXiv, arXiv:1904.10504.

[16] L. Chen, Y. Ye, T. Bourlai, Adversarial machine learning in malware detection: Arms race between evasion attack and defense, in: 2017 European Intelligence and Security Informatics Conference, EISIC, 2017, pp. 99–106, http://dx.doi.org/10.1109/EISIC.2017.21.

[17] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, F. Roli, Adversarial malware binaries: Evading deep learning for malware detection in executables, in: 2018 26th European Signal Processing Conference, EUSIPCO, 2018, pp. 533–537, http://dx.doi.org/10.23919/EUSIPCO.2018.8553214.

[18] L. Jia, B. Tang, C. Wu, Z. Wang, Z. Jiang, Y. Lai, Y. Kang, N. Liu, J. Zhang, FuncFooler: A practical black-box attack against learning-based binary code similarity detection methods, 2022, http://dx.doi.org/10.48550/ARXIV.2208.14191, URL arXiv https://arxiv.org/abs/2208.14191.

[19] C. Yang, J. Xu, S. Liang, Y. Wu, Y. Wen, B. Zhang, D. Meng, DeepMal: maliciousness-preserving adversarial instruction learning against static malware detection, Cybersecurity 4 (2021) 16, http://dx.doi.org/10.1186/s42400-021-00079-5.

[20] R. Harang, E.M. Rudd, SOREL-20M: A large scale benchmark dataset for malicious PE detection, 2020, arXiv:2012.07634.

[21] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, G. Wang, BODMAS: An open dataset for learning based temporal analysis of PE malware, in: 2021 IEEE Security and Privacy Workshops, SPW, 2021, pp. 78–84, http://dx.doi.org/10.1109/SPW53761.2021.00020.
[22] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, E. Weippl, Protecting software through obfuscation: Can it keep pace with progress in code analysis? ACM Comput. Surv. 49 (1) (2016) http://dx.doi.org/10.1145/2886012.
[23] C.S. Collberg, C.D. Thomborson, D. Low, Breaking abstractions and unstructuring data structures, in: Proceedings of the 1998 International Conference on Computer Languages (Cat. No.98CB36225), 1998, pp. 28–38.
[24] C. Nachenberg, Computer virus-antivirus coevolution, Commun. ACM 40 (1) (1997) 46–51, http://dx.doi.org/10.1145/242857.242869.
[25] L.M. Markus F.X.J. Oberhumer, J.F. Reiser, Ultimate packer for executables, 1996, https://upx.github.io/.
[26] O. Technologies, Themida overview, 2010, https://www.oreans.com/themida.php.
[27] S. King, P. Chen, SubVirt: implementing malware with virtual machines, in: 2006 IEEE Symposium on Security and Privacy (S&P'06), 2006, pp. 14–327, http://dx.doi.org/10.1109/SP.2006.38.
[28] P. Junod, J. Rinaldini, J. Wehrli, J. Michielin, Obfuscator-LLVM–software protection for the masses, in: 2015 IEEE/ACM 1st International Workshop on Software Protection, IEEE, 2015, pp. 3–9.
[29] M. Ollivier, S. Bardin, R. Bonichon, J.-Y. Marion, How to kill symbolic deobfuscation for free (or: Unleashing the potential of path-oriented protections), in: Proceedings of the 35th Annual Computer Security Applications Conference, ACSAC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 177–189, http://dx.doi.org/10.1145/3359789.3359812.
[30] VirusShare, Virusshare, 2023, https://virusshare.com/.
[31] Quarkslab, LIEF: library for instrumenting executable files, 2017–2018, https://lief.quarkslab.com/.
[32] O. Technologies, An interactive decompiler, disassembler, debugger, 2015, https://binary.ninja/.
[33] Libre reversing framework for unix geeks, 2013, https://github.com/radareorg/radare2.
[34] A powerful disassembler and a versatile debugger, 2012, https://hex-rays.com/IDA-pro/.
[35] The LLVM compiler infrastructure, 2008, https://llvm.org/.
[36] A dynamic binary instrumentation tool, 2010, https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html.
[37] An open-source binary analysis platform, 2018, https://angr.io/.
[38] A professional app shielding and hardening solution, 2017, https://www.preemptive.com/.
[39] An free, open-source protector for net applications, 2015, https://mkaring.github.io/ConfuserEx/.
[40] A multifunctional EXE packing tool, 2010, http://www.aspack.com/asprotect32.html.
[41] A professional system for executable files licensing and protection, 2012, https://www.enigmaprotector.com/.
[42] A tool protect program from being reversed, 2010, https://shell.virbox.com/.
[43] VMProtect protects code by executing it on a virtual machine, 2012, https://vmpsoft.com/.
[44] D. Hendrycks, T. Dietterich, Benchmarking neural network robustness to common corruptions and perturbations, in: Proceedings of the International Conference on Learning Representations, 2019.
[45] Machine learning static evasion competition 2019, 2019, https://github.com/endgameinc/malware_evasion_competition.
[46] The best antivirus protection, 2020, https://www.pcmag.com/picks/thebest-antivirus-protection.