# Towards A Framework for Preprocessing Analysis of Adversarial Windows Malware

Nicholas Schultz
*United States Military Academy*
West Point, NY, USA
nicholas.schultz@westpoint.edu

Adam Duby
*United States Military Academy*
West Point, NY, USA
adam.duby@westpoint.edu

2022 10th International Symposium on Digital Forensics and Security (ISDFS) | 978-1-6654-9796-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISDFS55398.2022.9800812

***Abstract*—Machine learning for malware detection and classification has shown promising results. However, motivated adversaries can thwart such classifiers by perturbing the classifier's input features. Feature perturbation can be realized by transforming the malware, inducing an adversarial drift in the problem space. Realizable adversarial malware is constrained by available software transformations that preserve the malware's original semantics yet perturb its features enough to cross a classifier's decision boundary. Further, transformations should be plausible and robust to preprocessing. If a defender can identify and filter the adversarial noise, then the utility of the adversarial approach is decreased. In this paper, we examine common adversarial techniques against a set of constraints that expose each technique's realizability. Our observations indicate that most adversarial perturbations can be reduced through forensic preprocessing of the malware, highlighting the advantage of forensic analysis prior to classification.**

***Index Terms*—Adversarial examples, malware analysis, malware forensics**

## I. Introduction

Machine learning (ML) and deep learning (DL) classifiers promise scale and accuracy in malware detection and familial classification. Classifiers such as Neurlux [1] and MalConv [2] have demonstrated high accuracy, saving analysts valuable time during incident response.

Adversarial malware poses a threat to these classifiers [3], [4]. Specifically designed to confuse malware classifiers, adversarial malware attempts to perturb or obfuscate their features at test time. By targeting specific features without modifying the semantics of the malware, adversarial malware reduces the utility of ML and DL classifiers without compromising the effectiveness of the malware.

In this paper, we adopt the formalization of adversarial constraints proposed by Pierazzi et al. [4] to analyze adversarial malware techniques from a malware forensic perspective. Specifically, we assess adversarial techniques for transformation availability, semantic preservation, plausibility, and robustness to preprocessing. We offer a general defensive framework for preprocessing adversarial malware based on malware forensics domain knowledge. Our findings show that many adversarial attacks can be identified by manual inspection, violating the constraint of plausibility. Further, we observe that the adversarial perturbations can be reduced by preprocessing the malware before feature extraction.

The benefits of our observations are two-fold: (i) advances in adversarial research should focus on attacking features in a manner that is difficult to identify (i.e., plausible) and remove through forensic analysis (i.e., robust to preprocessing); and (ii) defenders can incorporate a forensic preprocessing pipeline to reduce the effects of adversarial perturbations. Specifically, this paper makes the following contributions:

- Propose forensic preprocessing in the machine learning pipeline to filter adversarial noise before feature extraction;
- Analyze existing adversarial Windows malware techniques against the framework of constraints proposed by Pierazzi et al. [4].

The rest of the paper is organized as follows. Section II provides an overview of malware classification and existing adversarial techniques. In Section III, we analyze adversarial techniques against a set of constraints, and present forensic preprocessing techniques to filter adversarial noise to reduce the adversarial feature perturbation. We conclude with Section IV.

## II. Background and Related Work

This paper focuses on forensic analysis of malware features that an attacker can perturb to thwart a classifier. The classifier input are features extracted from the malware, and the classifier's response variable is selected from the label space. These classifiers are commonly represented as $f : X \rightarrow Y$, where $X$ is the feature space and $Y$ is the label space. Given a feature vector $x \in X$, the classifier returns a target label, $f(x) = y$, where $y \in Y$. The label space varies with respect to the classifier's goal. For example, some classifiers detect malware. As such, their label space is binary: malicious or benign. By contrast, malware family classifiers label malware by family. Such classifiers have a multiclass label space and they often assume the input is already malicious.

### A. Feature Selection

For each $m \in M$ where $M$ is the problem space of PE32 malware files, a feature extraction function produces the feature vector, $\phi(m) = x$, where $x \in X$. Choosing the optimal features is an open area of research. Related work has shown

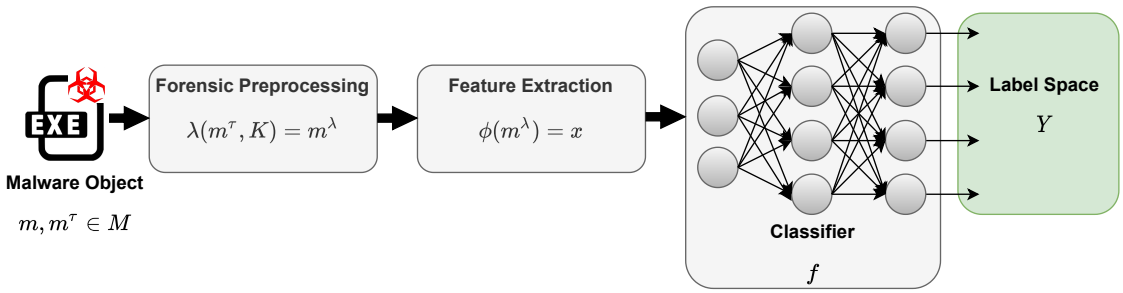


Fig. 1: Feature engineering pipeline.

some promising results in static features. Despite their success, packing and obfuscation limit their utility in malware analysis [5]. By contrast, dynamic features overcome the limitations of static analysis by extracting information from the running process. Dynamic feature extraction incurs additional time and complexity to run the malware.

### B. Adversarial Malware

Adversarial malware transforms the malware at test time to frustrate the classifier. Given a malware object $m \in M$ whose label is $y \in Y$, the attacker applies a transformation function $\tau(m) = m'$, where $\tau \in \mathcal{T}$, $m'$ is the adversarial variant of $m$, and $\phi(m) \neq \phi(m')$. The adversarial transformation is successful if the classifier yields a response such that $f(m) \neq f(m')$. In other words, $f(m') = y'$, where $y' \neq y$. Since the classifier input is from the feature space, the transformation of $m$ must perturb the feature vectors enough to cross the classifier's decision boundary. Therefore, the transformation must produce a perturbation vector, defined as $\delta = x' - x$. To optimize $\delta$, the adversary may apply a series of transformations on $m$, where $\tau(m) = \tau_n \circ \tau_{n-1} \circ ... \circ \tau_1$. In the following section, we present related work on popular adversarial transformations.

### C. Adversarial Approaches

*Perturb Header Files:* Adversaries can transform the bytes of the malware's PE32 header [6]. This approach relies on byte level modification of the header, making it difficult to reverse engineer the original malware [6], [7]. However, perturbation of header files is difficult to control, and often leads to unintended changes to the malware's functionality [7].

*Filling Slack Space:* Slack space attacks modify the unused bytes of a header file and add noise in the unused space between the PE32 file sections. The executable remains unaffected while the properties of the file change, potentially confusing classifiers [8], [9].

*Padding:* In contrast to changing the header and modifying slack space, padding adds superfluous bytes to the end of a file, changing the file's size and byte distribution without affecting the program's semantics [8], [10].

*Code Substitution:* The more sophisticated code substitution techniques rewrite the malware code at the assembly level, replacing the original assembly instructions with different, yet semantically equivalent, instructions. As such, the functionality of the malware is maintained, but the byte-level features are different from the original malware [11].

*Full Translation:* Rather than replacing the code directly, a full translation approach lifts the assembly to a higher language and reassembles the code with perturbations. Akin to code substitution, this yields a new assembly with the same functionality as the original malware [12].

*Payload Encoding:* By encoding parts of the malware, often through XOR or Base64 encoding, adversaries can conceal many features until run-time. The malware is decoded when an embedded decoder deciphers the code for execution [13].

*Packing:* Packing is an approach that compresses the malware, reducing its size and significantly altering its properties. This approach makes reverse engineering and static analysis particularly difficult, as the malware file is obfuscated until the file is unpacked and executed at run-time [5], [14], [15].

*Section Injection:* The section table of a malware file contains all the metadata for its respective sections necessary for malware to be compiled and executed. The section injection approach inserts an additional entry into the section table, providing a new section foreign to the original malware. This enables the adversary to inject new content and significantly perturb the features accordingly [16].

*IAT Injection:* The Import Address Table (IAT) of a file contains all of the files imports and library calls. the IAT injection approach attacks this space, importing excess functions and manipulating the library calls of the executable [17]. This both extends the IAT and provides an avenue to import custom libraries that add perturbations to the malware [18].

*Dropper:* To use the dropper adversarial approach, the adversary embeds the original malware in the resource section of a new executable. The new executable appears to be non-nefarious, as its only purpose is to extract the embedded malware and execute it at run-time. This effectively combats classifiers trained on file properties and classifiers trained on API calls, as the dropper conceals these features behind the new executable [19].

### D. Adversarial Constraints in the Problem Space

Adversaries face unique challenges when attacking the feature space that make crafting realistic adversarial examples a challenge. For an attack to be realizable, the adversary must transform the problem space object in a manner that effects

the feature space. It is trivial to conceive perturbations in the feature space, but the realizability of such perturbations is more complex. Pierazzi et al. formalized this observation and propose four constraints on the adversary to make attacks realizable [4]:

1) **Available Transformations**: A transformation function $\tau$ must exist that perturbs the targeted feature space;
2) **Preserved Semantics**: $\tau$ must not hinder the desired semantics of the malware;
3) **Plausibility**: The characteristics of the transformed malware must appear natural and consistent with similar malware;
4) **Robustness to Preprocessing**: $\tau$ should be robust against filtering or preprocessing that negates the perturbation vector.

In Section III-C, we assess popular adversarial techniques against these constraints.

## III. Forensic Preprocessing

In this section, we present a theoretical framework for preprocessing malware samples before feature extraction in order to reduce the effects of adversarial perturbations.

### A. Threat Model

We assume the attacker has perfect knowledge of the target classifier's feature space. As such, the attacker knows the feature extraction function $\phi(m)$. Indeed, defenders may deploy classifiers that are trained using a feature space that is orthogonal to the perturbed features. However, our work focuses on reducing the effects of an attack that targets the classifier's feature space.

### B. Overview of Theorem

As discussed in Section II-A, the adversary applies a series of transformations on the malware that perturbs the feature space. We hypothesize that for each transformation, there exists some forensic preprocessing function $\lambda \in \Lambda$ that reduces the adversarial perturbation vector $\delta$. More formally:

$$\begin{aligned} \forall \tau \in \mathcal{T} : \phi(\tau(m)) = x + \delta, \\ \exists \lambda \in \Lambda : \phi(\lambda(m^\tau)) = x + \delta^\lambda, \delta^\lambda \ll \delta \end{aligned} \tag{1}$$

Due to potential *side effect* perturbations induced by the transformation [4], we cannot claim that $\lambda(m^\tau) = \tau^{-1}(m)$. In other words, the goal of forensic preprocessing is not to compute the inverse of the transformation function, but to reduce the size of the perturbation vector.

The preprocessing step applies domain knowledge $K$ about the problem space to select the appropriate $\lambda$ from the set of all possible preprocessing functions $\Lambda$. As such, we say $\lambda(m^\tau, K) = m^\lambda$, where $\phi(m^\lambda) \approx \phi(m)$. Since the adversary may choose a series of transformations, the defender may need to apply a series of preprocessing operators to counter each transformation. Forensic preprocessing introduces a new stage in the feature engineering pipeline, as shown in Fig. 1.

### C. Analyzing Adversarial Transformations for Constraints

We examine the popular adversarial transformations with respect to the constraints discussed in Section II-D. Our findings are shown in Table I. Although many transformations are indeed available, they do not necessarily meet the remaining constraints discussed in Section II-D. From an attacker perspective, the most important constraint is semantic preservation. If the perturbation reduces the malware's effectiveness, then the adversary will not be able to achieve its desired effects on the target. For example, random perturbations of bytes in the PE32 header does not always guarantee that the malware will function as desired [6], [7]. Only certain bytes can be altered without breaking the program's functionality, which reduces the attack surface of the header.

Many techniques can be identified through manual inspection, violating the constraint of plausibility. For example, inserting adversarial noise and padding can be identified through manual inspection of the binary file. Attackers can insert noise in the unused slack space in the header and in slack space between PE32 sections. As shown in Fig. 2, manual inspection shows what a normal header and file layout looks like that is consistent with other files. We also observe that almost every assessed technique is susceptible to preprocessing. This offers defenders the opportunity to reduce the effects of adversarial perturbations by filtering out the adversarial noise.

### D. Forensic Preprocessing Examples

In this section we discuss preprocessing functions on several adversarial techniques that reduce the perturbation vector.

*1) PE Header Perturbation:* Perturbations of the PE Header are a difficult adversarial approach to pursue because it impacts the functionality of the malware and often completely corrupts the file. By affecting the malware's functionality, this violates the semantic preservation constraint, making it a low threat consideration for preprocessing. Despite this, certain methods can limit the negative effects of perturbation on malware, and preserve the semantics to a limited degree [6]. Perturbing the PE Header reduces an analyst's ability to reverse engineer the malware without an original file to reference. However, by analyzing the raw binary and reconstructing header values from the information available, it is possible to extract features from the perturbed file. While much of the information will still be missing, applying default values of certain fields and using heuristics to extract additional information will provide a partially reconstructed PE Header file worthy of analysis [20].

*2) Slack Space and Padding:* Malware classifiers trained using header and byte level features can be attacked through replacing the empty bytes at the end of each section or padding the end of the file with superfluous data. These approaches are cheap and feasible, as the adversary can extract the PE Header information and identify where to add or manipulate the data. For slack attacks, the adversary can extract the `RawAddress, RawSize`, and `VirtualSize` of each section with simple PE Header analysis. With this information, the adversary can compute the exact slack space between sections

TABLE I: Assessment of Adversarial Approaches

| Adversarial Approach | Available Transformations | Preserved Semantics | Plausibility | Robust to Preprocessing |
|---|---|---|---|---|
| *Perturb Header* [6], [7] | ✓ | ✗ | ✓ | ✗ [20] |
| *Fill Slack Space* [8], [9] | ✓ | ✓ | ✗ | ✗ |
| *Padding* [8], [10] | ✓ | ✓ | ✗ | ✗ |
| *Translation* [12], [21] | ✓ | ✓ | ✓ | ✗ [22], [23] |
| *Code Substitution* [11] | ✓ | ✓ | ✓ | ✗ [22], [23] |
| *Encoding* [13] | ✓ | ✓ | ✗ | ✗ |
| *Packing* [5] | ✓ | ✓ | ✗ | ✗ [24], [25] |
| *Section Injection* [16] | ✓ | ✓ | ✓ | ✗ |
| *IAT Injection* [17], [18] | ✓ | ✓ | ✓ | ✗ |
| *Dropper* [19] | ✓ | ✓ | ✓ | ✗ |

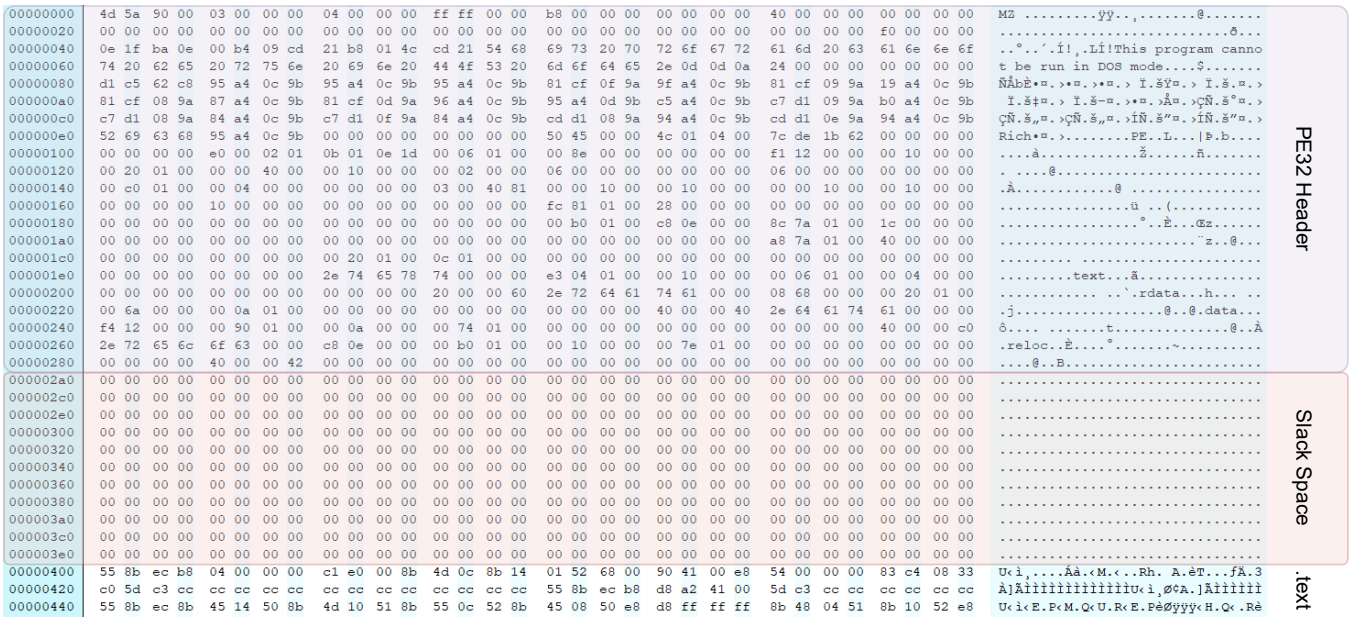


Fig. 2: PE32 slack space.

by adding `RawAddress` and `VirtualSize` to determine the start of the slack space, and using the next section's `RawAddress` to determine the end of the slack space for that section. Thus, for there to be an available transformation, the $\tau$ must simply replace the empty bytes in this computed space and the file will be perturbed.

Since the executable will never use these bytes, this approach successfully avoids disrupting the functionality of the malware, thus preserving its semantics. However, a forensic analyst can easily identify the slack space and observe any perturbations since the slack space are visually apparent as shown in Fig. 2. Accordingly, this approach falls short of the plausibility constraint. Similarly, just as the transformation $\tau$ is cheap and simple for the adversary, it is easy to preprocess through a rudimentary function $\lambda(m^{\tau})$ that identifies the slack space through the same process and replaces the random bytes with empty bytes. The process for preprocessing slack space is as follows:

```
def CleanSlack:
    for section in peHeader:
        slackStart = RawAddress + VirtualSize
        slackEnd = nextRawAddress - 1
        Insert_Zero_Bytes(slackStart,slackEnd)
```

Padding attacks append superfluous data to the end of the file, maintaining the semantics of the file while changing its physical properties. However, this approach also lacks plausibility, as a forensic analyst can observe the added data and preprocess the file through cutting the appended section off of the file, thus removing the perturbation vector.

*3) Code Substitution and Translation:* The assembly code of an executable can be extracted and analyzed for malicious patterns by classifiers [26]. However, through code substitution and translation, adversaries can introduce malware with the same semantics and different code than these classifiers train on. This transformation is accomplished through programs that find common assembly instructions and replace them with equivalent ones, or lift the code to a higher language and retranslate it back into assembly. While the assembly code

changes, the semantics are preserved. Assuming a forensic analyst does not have access to the original malware file, this approach maintains plausibility since it is not possible to know the original code. Despite this, there is a proposed preprocessing method to subvert this approach. Just as the adversary lifts the code to a higher language to obfuscate the assembly, analysts may lift the training set of malware. In this way, the classifier is trained on a consistent translation method. Adversarial malware at test-time can then be lifted using, normalizing the feature space prior to feature extraction [22], [23].

*4) Packing and Encoding:* Packing and encryption are simple methods used to thwart nearly any classifier relying on static features without preprocessing. The transformations are easy: compress the malware or encode it with XOR or Base64 encoding so defenders have to execute the malware to extract the features. Compression and encoding of malware does not affect the semantics, so it is a feasible adversarial approach to perturbation. Since dynamic analysis is more expensive in time and complexity than static analysis, preprocessing this approach is ideal to maintain the accuracy of classifiers training on static features. To preprocess packed malware, numerous unpackers can be employed to decompress the malware into its original state [24], [25]. For encoded malware, forensic analysts can identify the encoding method and decode it appropriately prior to feature extractions [27]–[29].

*5) IAT Injection:* Features extracted from the IAT can be perturbed by adding superfluous libraries (DLLs) and function calls. This can be realized using the Python LIEF library[1]. Specifically, the adversary can add libraries to the PE32's `IMAGE_IMPORT_DESCRIPTOR` and imported function entries can be added to the import name table (INT) and import address table (IAT).

Adversarial noise in the import feature are never actually invoked. As such, import related features can be preprocessed using IDA[2] to check for cross references (XREFs). If the superfluous import is never referenced, it can be excluded from the feature vector. Analysts can also parse the `IMAGE_IMPORT_DESCRIPTOR` for XREF addresses and identify noisy imports:

```
def findImports:
    for entry in IMAGE_DIRECTORY_IMPORT:
        for function in entry.imports:
            find function XREF
```

Adversaries can also remove entries by manually delaying loading the imports, as shown below. In this example, both the Win32 `MessageBoxA` API and its associated DLL `user32.dll` will be neglected from static import features. However, forensic preprocessing can identify the presence of this technique through the presence of `LoadLibrary` and `GetProcAddress`. The arguments from these functions are extracted and added to the import features.

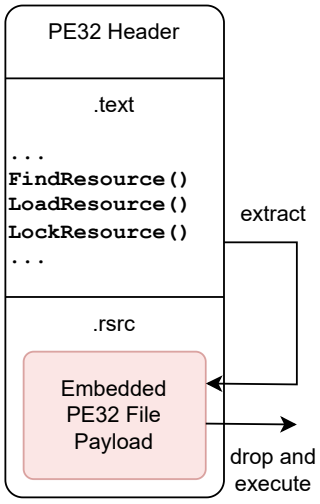


Fig. 3: Malware dropper.

```
typedef int (*foo)(void);
foo _foo;
HINSTANCE h = LoadLibrary("user32.dll");
_foo = (foo)GetProcAddress(h, "MessageBoxA");
_foo();
```

*6) Dropper:* Embedding a second stage payload within a dropper file is a popular malware technique. Further, it drastically perturbs the features of any malware variant. The most common techinque is to embed the original malware in the resource section (.rsrc) of the dropper and extract it at runtime, as shown in Figure 3.

Analysts can identify the technique by either recognizing the API calls that interact with the resource section shown in Figure 3, or by the existence of PE32 file signatures embedded within the file. If the embedded file is encoded, then an additional preprocessing technique can decode the file. Analysts can also take advantage of Mandiant's CAPA tool[3] that can detect this technique. Once identified, the embedded PE32 is carved from the dropper and feature extraction is performed on the embedded file instead of the host PE32 file.

## IV. Conclusion and Future Work

In this paper, we analyzed common adversarial transformations of Windows malware for their ability to avoid detection and preprocessing by a malware analyst. While adversarial malware poses a threat to machine learning classifiers, the risk of an adversarially induced misclassification can be reduced with forensic preprocessing using malware analysis domain knowledge. All of the adversarial approaches discussed in this paper are susceptible to some degree of preprocessing. However, these approaches are all considered independently; the effects of multiple approaches increase the complexity of

[1]https://github.com/lief-project/LIEF

[2]https://hex-rays.com/ida-pro/

[3]https://github.com/mandiant/capa

preprocessing and could present unanticipated impediments to classifiers.

Moving forward, adversarial researchers should consider techniques that are difficult to preprocess, while defenders should select features that are difficult for adversaries to perturb. However, the importance of the constraints proposed by Pierazzi et al. [4] remain significant for adversarial approaches: if a transformation is feasible without compromising the semantics of the malware or alerting a forensic analyst, then the adversarial malware is realizable. But if the approach lacks robustness to preprocessing, then its perturbation vector can be reduced through preprocessing methods.

The robustness to preprocessing constraint proposed by Pierazzi et al. [4] can be ignored by the adversary if defenders fail to conduct proper preprocessing methods. Therefore, we recommend future work focus on automated employment of these preprocessing techniques prior to feature extraction to improve classifier robustness against adversarial drift. With the automated implementation of preprocessing methods, adversarial malware will have to meet all constraints to prove a realizable threat to modern malware classifiers.

## ACKNOWLEDGMENT

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the United States Military Academy, the United States Army, the Department of Defense, or the United States Government.

## REFERENCES

[1] C. Jindal, C. Salls, H. Aghakhani, K. Long, C. Kruegel, and G. Vigna, "Neurlux: dynamic malware analysis without feature engineering," in *35th ACM Annual Computer Security Applications Conference (ACSAC)*, 2019.

[2] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] D. Li, Q. Li, Y. Ye, and S. Xu, "Arms race in adversarial malware detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–35, 2021.

[4] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *41st IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 1332–1349.

[5] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When malware is packin'heat; limits of machine learning classifiers based on static analysis features," in *27th Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[6] R. L. Castro, C. Schmitt, and 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)*. IEEE, 2019, pp. 240–247.

[7] R. L. Castro, C. Schmitt, and G. D. Rodosek, "Armed: How automatic malware modifications can evade static detection?" in *2019 5th International Conference on Information Management (ICIM)*. IEEE, 2019, pp. 20–27.

[8] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019.

[9] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," *arXiv preprint arXiv:1802.04528*, 2018.

[10] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European signal processing conference (EUSIPCO)*. IEEE, 2018, pp. 533–537.

[11] E. Konstantinou and S. Wolthusen, "Metamorphic virus: Analysis and detection," *Royal Holloway University of London*, vol. 15, p. 15, 2008.

[12] M. Wenzl, G. Merzdovnik, J. Ullrich, and E. Weippl, "From hack to elaborate technique—a survey on binary rewriting," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1–37, 2019.

[13] F. Ceschin, M. Botacin, G. Lüders, H. M. Gomes, L. Oliveira, and A. Gregio, "No need to teach new tricks to old malware: Winning an evasion challenge with xor-based adversarial samples," in *ACM Reversing and Offensive-oriented Trends Symposium*, 2020, pp. 13–22.

[14] J. Singh and J. Singh, "Challenge of malware analysis: malware obfuscation techniques," *International Journal of Information Security Science*, vol. 7, no. 3, pp. 100–110, 2018.

[15] A. Mantovani, S. Aonzo, X. Ugarte-Pedrero, A. Merlo, and D. Balzarotti, "Prevalence and impact of low-entropy packing schemes in the malware ecosystem." in *NDSS*, 2020.

[16] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Explaining vulnerabilities of deep learning to adversarial malware binaries," in *3rd Italian Conference on Cyber Security (ITASEC)*, 2019.

[17] D. Korczynski, "Repeconstruct: reconstructing binaries with self-modifying code and import address table destruction," in *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2016, pp. 1–8.

[18] B. Mariani, "Userland hooking in windows," *by High-Tech Bridge SA*, vol. 33, 2011.

[19] F. Ceschin, M. Botacin, H. M. Gomes, L. S. Oliveira, and A. Grégio, "Shallow security: On the creation of adversarial variants to evade machine learning-based malware detectors," in *Proceedings of the 3rd Reversing and Offensive-oriented Trends Symposium*, 2019, pp. 1–9.

[20] X. Hu, J. Jang, T. Wang, Z. Ashraf, M. P. Stoecklin, and D. Kirat, "Scalable malware classification with multifaceted content features and threat intelligence," *IBM Journal of Research and Development*, vol. 60, no. 4, pp. 6–1, 2016.

[21] D. Chisnall, "The challenge of cross-language interoperability," *Communications of the ACM*, vol. 56, no. 12, pp. 50–56, 2013.

[22] T. Kroes, A. Altinay, J. Nash, Y. Na, S. Volckaert, H. Bos, M. Franz, and C. Giuffrida, "Binrec: Attack surface reduction through dynamic binary recovery," in *Proceedings of the 2018 Workshop on Forming an Ecosystem Around Software Transformation*, 2018, pp. 8–13.

[23] A. Oikonomopoulos, R. Vermeulen, C. Giuffrida, and H. Bos, "On the effectiveness of code normalization for function identification," in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2018, pp. 241–251.

[24] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee, "Polyunpack: Automating the hidden-code extraction of unpack-executing malware," in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE, 2006, pp. 289–300.

[25] K. Hajarnis, J. Dalal, R. Bawale, J. Abraham, and A. Matange, "A comprehensive solution for obfuscation detection and removal based on comparative analysis of deobfuscation tools," in *2021 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*. IEEE, 2021, pp. 1–7.

[26] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.

[27] K. Coogan, S. Debray, T. Kaochar, and G. Townsend, "Automatic static unpacking of malware binaries," in *2009 16th Working Conference on Reverse Engineering*. IEEE, 2009, pp. 167–176.

[28] J. Raphel and P. Vinod, "Information theoretic method for classification of packed and encoded files," in *Proceedings of the 8th International Conference on Security of Information and Networks*, 2015, pp. 296–303.

[29] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda, "Inspector gadget: Automated extraction of proprietary gadgets from malware binaries," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 29–44.