Towards Adversarial Malware Detection: Lessons Learned from PDF-based Attacks

DAVIDE MAIORCA, University of Cagliari
BATTISTA BIGGIO and GIORGIO GIACINTO, University of Cagliari and Pluribus One, Italy

Malware still constitutes a major threat in the cybersecurity landscape, also due to the widespread use of infection vectors such as documents. These infection vectors hide embedded malicious code to the victim users, facilitating the use of social engineering techniques to infect their machines. Research showed that machine-learning algorithms provide effective detection mechanisms against such threats, but the existence of an arms race in adversarial settings has recently challenged such systems. In this work, we focus on malware embedded in PDF files as a representative case of such an arms race. We start by providing a comprehensive taxonomy of the different approaches used to generate PDF malware and of the corresponding learning-based detection systems. We then categorize threats specifically targeted against learning-based PDF malware detectors using a well-established framework in the field of adversarial machine learning. This framework allows us to categorize known vulnerabilities of learning-based PDF malware detectors and to identify novel attacks that may threaten such systems, along with the potential defense mechanisms that can mitigate the impact of such threats. We conclude the article by discussing how such findings highlight promising research directions towards tackling the more general challenge of designing robust malware detectors in adversarial settings.

CCS Concepts: • **Security and privacy → Malware and its mitigation**; • **Theory of computation → Adversarial learning**;

Additional Key Words and Phrases: PDF files, infection vectors, machine learning, evasion attacks, vulnerabilities, javascript

ACM Reference format:
Davide Maiorca, Battista Biggio, and Giorgio Giacinto. 2019. Towards Adversarial Malware Detection: Lessons Learned from PDF-based Attacks. *ACM Comput. Surv.* 52, 4, Article 78 (August 2019), 36 pages.
https://doi.org/10.1145/3332184

This work was partially supported by the INCLOSEC (funded by Sardegna Ricerche—CUP G88C1700 0080006) and PISDAS (funded by Regione Autonoma della Sardegna—CUP E27H14003150007) projects; and by the EU H2020 project ALOHA, under the European Union's Horizon 2020 research and innovation programme (grant no. 780788).
Authors' addresses: D. Maiorca, University of Cagliari, Piazza d'Armi, Cagliari, 09123, Italy; email: davide.maiorca@unica.it; B. Biggio and G. Giacinto, University of Cagliari, Piazza d'Armi, Cagliari, 09123, Italy, Pluribus One, Italy; emails: battista.biggio@diee.unica.it, giacinto@unica.it.

0360-0300/2019/08-ART78 $15.00
https://doi.org/10.1145/3332184

1 INTRODUCTION

Malware for X86 (and more recently for mobile architectures) is still considered one of the top threats in computer security. While it is common to think that the most dangerous attacks are crafted using plain executable files (especially on Windows-based operating systems), security reports showed that the most dangerous attacks in the wild were carried out by using *infection vectors* [87]. With this term, we define non-executable files whose aim is exploiting vulnerabilities of third-party applications to trigger download (or direct execution) of executable payloads. Using such vectors gives attackers multiple advantages. First, they can exploit the structure of third-party formats to conceal malicious code, making their detection significantly harder. Second, infection vectors can be effectively used in social engineering campaigns, as victims are more prone to receive and open documents or multimedia content. Finally, although vulnerabilities of third-party applications are often publicly disclosed, they are not promptly patched. The absence of proper security updates thus makes the lifespan of attacks perpetrated with infection vectors much longer.

Machine learning–based technologies have been increasingly used both in academic and industrial environments (see, e.g., Reference [47]) to detect malware embedded in infection vectors like malicious PDF files. Research work has demonstrated that learning-based systems could be effective at detecting obfuscated attacks that are typically able to evade simple heuristics [22, 64, 82, 95], but the problem is still far from being solved. Despite the significant increment of detected attacks, researchers started questioning the reliability of learning algorithms against *adversarial* attacks carefully crafted against them [7–9, 16, 17]. Such attacks became widely popular when researchers showed that it was possible to evade deep learning algorithms for computer vision with *adversarial examples*, i.e., minimally perturbed images that mislead classification [39, 88]. The same attack principles have also been employed to craft *adversarial malware samples*, as first shown in Reference [8], and subsequently explored in References [28, 41, 51, 96, 99]. Such attacks can typically perform few, fine-grained changes on correctly detected, malicious samples to have them misclassified as legitimate. Accordingly, it becomes possible to evade machine-learning detection in a stealthier manner without resorting to invasive changes like those performed through code obfuscation.

Malicious PDF files constitute the most-studied infection vectors in adversarial environments [8, 19, 60–63, 82, 83, 96, 102]. This file type was chosen for three main reasons: First, it represented the most dangerous infection vector in the wild from 2010 to 2014 (to be subsequently replaced by Office-based malware), and a lot of machine learning–based systems were developed to detect the vast variety of polymorphic attacks related to such a format (e.g., References [64, 82, 83]). Second, the complexity of the PDF file format allows attackers to employ various solutions to conceal code injection or other attack strategies. Finally, it is easy to modify its structure by for example injecting benign or malicious material in various portions of the file. Such a characteristic makes PDF files particularly prone to be used in adversarial environments, as the effort that attackers have to carry out to create attack samples is significantly low.

While previous surveys in the field of PDF malware analysis focused on describing the properties of detection systems [31, 71], our work explores the topic under the perspective of adversarial machine learning. The idea is showing how adversarial attacks have been carried out against PDF malware detectors by exploiting the vulnerabilities of their essential components and highlighting how the arms race between attackers and defenders has evolved in this scenario over the past decade. The result is two-fold: on the one hand, we highlight the current security issues that allow attackers to deceive the current state-of-the-art algorithms; on the other hand, understanding adversarial attacks points out new, intriguing research directions that we believe can also be relevant for other malware detection problems.

To adequately describe PDF malware detection under adversarial environments, we organized our work as follows: First, we describe the main attack types that can be carried out by using PDF files. Second, we provide a detailed description of state-of-the-art PDF detectors by primarily focusing on their machine-learning components. Third, we show how such systems can be evaded with different adversarial attacks. In particular, we provide a complete taxonomy of the attacks that can be performed against learning-based detectors by also describing how such attacks can be concretely implemented and deployed. Finally, we overview possible solutions that have been proposed to mitigate such attacks, thus sketching further research directions in developing novel adversarial attacks and defenses against them.

This work also aims to provide solid bases to overcome the hurdles that can be encountered when working in adversarial environments. We firmly believe that these principles can constitute useful guidelines when dealing with the generic task of malware detection, not only restricted to PDF files. We claim that systems based on machine learning for malware detection should be built by accounting for the presence of attacks carefully tailored against them, i.e., according to the principle of *security by design*, which suggests to proactively model and anticipate the attacker to build more secure systems [13, 16].

The rest of the article is structured as follows: Section 2 provides a general overview of the attacks carried out with the PDF file format, as well as the possible attacks in the wild that can be carried out against PDF readers; Section 3 depicts the various detection methodologies that have been introduced in these years by also discussing the achieved detection performances; Section 4 provides a complete taxonomy of the adversarial attacks that can be carried out by exploiting learning-based vulnerabilities, as well as an insight into how these attacks can be implemented; Section 5 describes the current countermeasures adopted against adversarial threats and sketches possible research directions for developing new attacks and defenses. Section 6 concludes the article by also providing a comparison with previous works.

2 PDF MALWARE

This section aims to provide the basics to understand how PDF malware in the wild executes its malicious actions. To this end, we divided this section into two parts. In the first part (Section 2.1), we provide a comprehensive overview of the PDF file format. In the second part (Section 2.2), we describe how PDF malware uses the characteristics of its file format to exploit vulnerabilities.

2.1 Overview of PDF Files

Digital documents—albeit different concerning the way their readers parse them—share some essential aspects. In particular, they can be typically represented as a combination of two components: a general *structure* that outlines how the document contents are stored, and the *file content* that describes the information that is properly visualized to the user (such as text, images, scripting code).

General Structure. PDF (Portable Document Format) is one of the most-used formats, along with Microsoft Office, to render digital documents. It can be conceptually considered as a graph of objects, each of them performing specific actions (e.g., displaying text, rendering images, executing code, and so forth). The typical structure of a PDF file is showed in Figure 1, and it consists of four parts [1, 3]:

- **Header.** A single line of text containing information about the PDF file version, introduced by the marker %.
- **Body.** A sequence of objects that define the operations performed by the file. Such objects can also contain compressed or uncompressed embedded data (e.g., text, images, scripting

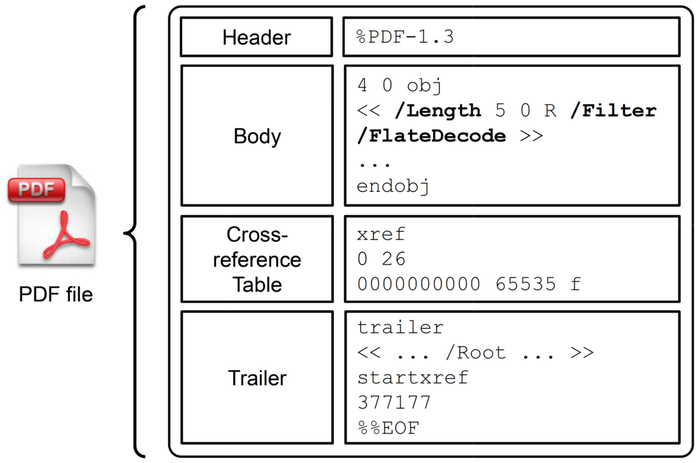

Fig. 1. PDF file structure, with examples of header, body, cross-reference table, and trailer contents. Object names (i.e., keywords) are highlighted in bold.

code). Each object possesses a unique reference number, typically introduced by the sequence `number 0 obj`,[1] where `number` is the proper object number. PDF objects can be referenced by others by using the sequence `number 0 R`,[2] where `number` identifies the *target* object that is referenced. Each object is ended by the `endobj` marker. The functionality of each object is described by *keywords* (also known as *name objects*—highlighted in bold in the figure), which are typically introduced by `/`.

- **Cross-reference (X-Ref) Table.** A list of offsets that indicate the position of each object in the file. Such a list gives the PDF readers precise indications on where to begin parsing each object. The cross-reference Table is introduced by the marker `xref`, followed by a sequence of numbers, whose last indicates the total number of objects in the file. Each line of the table corresponds to a specific object, but only lines that end with `n` are related to objects that are concretely stored in the file. It is worth noting that all the PDF readers parse *only the objects that are referenced by the X-Ref Table*. Therefore, it is possible to find objects that are stored in the file but that lack their reference in the table.
- **Trailer.** A special object that describes some basic elements of the file, such as the first object of the graph (i.e., where the PDF readers should start parsing the file information). Moreover, it contains references to the file metadata, which are typically stored in one single object. The keyword `trailer` always introduces the trailer object.

PDF files are parsed as follows: first, the trailer object is parsed by PDF readers to retrieve the first object of the hierarchy. Then, each object of the PDF graph (contained in the body) is explored and rendered by combining information contained in the X-Ref Table with the references numbers found inside each object. Every PDF file is terminated by a `%%EOF` (End Of File) marker. An interesting characteristic of PDF files is that they can also be *updated* without being regenerated from scratch (although the latter is possible), with a process called *versioning*. When an existing object is modified or new objects are added to the file, a new body, X-Ref table, and trailer are

[1] The value between `number` and `obj` is called *generation number*, and it is typically 0. It can be different in some corner cases, but we recommend to refer to the official documentation for more information.
[2] The value between `number` and `R` is the same generation number as the original object.

appended to the file. The new body and X-Ref table only contain information about the changes that occurred to the document.

Objects. As previously stated, objects are typically identified by a number, and they are more formally referred to as *indirect objects*. However, every element inside the body is generally regarded as an object, even if a number does not identify it. When an object is not identified by a number (i.e., when its part of other objects), it is called *direct*.

Indirect objects are typically composed of a combination of direct objects. Listing 1 reports a typical example of a PDF object.

```
8 0 obj
<</Filter[/FlateDecode]/Length 384/Type/EmbeddedFile>>
stream
...
endstream
endobj
```

Listing 1. An example of a PDF object. The content of the stream has not been reported for the sake of brevity.

Most of the times, indirect objects (in this case, the object is number 8) are dictionaries (enclosed in << >>) that contain *sequences of coupled direct objects*. Each couple typically provides specific information about the object. The object in the Listing contains a sequence of three couples of objects, which decompress an embedded file (marked by the keywords `/Type/EmbeddedFile`) of length 384 (keyword `/Length`) with a FlateDecode compression filter (keywords `/Filter/FlateDecode`). Objects that contain streams always feature the markers `stream` and `endstream` at the very end, meaning that PDF objects first instruct the PDF readers about their functionality, then on the data type they operate.

Among the various types of objects, there are some that perform actions such as executing `JavaScript` code, opening embedded files, or even performing automatic actions when the file is opened or closed. To this end, the PDF language resorts to specific name objects (keywords) that are typically associated with actions that can have repercussions from a security perspective. Among the others, we mention `/JavaScript` and `/JS` for executing `JavaScript` code; `/Names`, `/OpenAction`, `/AA` for performing automatic actions; `/EmbeddedFile` and `/F` for opening embedded files; `/AcroForm` to execute forms. The presence of such objects should be considered as a first hint that somebody may perform malicious actions. However, as such objects are also widely used in benign files, it may be quite hard to establish the maliciousness of the file by only inspecting them.

2.2 Malicious Exploitation

The majority of attacks that are carried out using documents resort to scripting codes to execute malicious code. Therefore, after having described the general structure of PDF files, we now provide an insight into the security issues related to the contents that can be embedded in the file. To do so, we provide a taxonomy of the various attacks that can be perpetrated by using PDF files. In particular, we focus on attacks targeting Adobe Reader, the most-used PDF reader in the wild. The common idea behind all attacks is that they exploit specific vulnerabilities of the Adobe Reader components, and in particular its plugins. Vulnerabilities can be characterized by multiple exploitation strategies, which also depend on the targeted Reader component.

Table 1. An Overview of the Main Exploited Vulnerabilities Against Adobe Reader, Along with Their Vulnerability and Exploitation Type

Vulnerability	Vuln. Type	Exploitation Type
CVE-2008-0655	API Overflow (*Collab.collectEmailInfo*)	JavaScript
CVE-2008-2992	API Overflow (*util.printf*)	JavaScript
CVE-2009-0658	Overflow	File Embedding (JBIG2)
CVE-2009-0927	API Overflow (*Collab.getIcon*)	JavaScript
CVE-2009-1492	API Overflow (*getAnnots*)	JavaScript
CVE-2009-1862	Flash (Memory Corruption)	ActionScript
CVE-2009-3459	Malformed Data (FlateDecode Stream)	JavaScript
CVE-2009-3953	Malformed Data (U3D)	JavaScript
CVE-2009-4324	Use-After-free (*media.newPlayer*)	JavaScript
CVE-2010-0188	Overflow	File Embedding (TIFF)
CVE-2010-1240	Launch Action	File Embedding (EXE)
CVE-2010-1297	Flash (Memory Corruption)	ActionScript
CVE-2010-2883	Overflow (*coolType.dll*)	JavaScript
CVE-2010-2884	Flash (Memory Corruption)	ActionScript
CVE-2010-3654	Flash (Memory Corruption)	ActionScript
CVE-2011-0609	Flash (Bytecode Verification)	ActionScript
CVE-2011-0611	Flash (Memory Corruption)	ActionScript
CVE-2011-2462	Malformed U3D Data	JavaScript
CVE-2011-4369	Corrupted PRC Component	JavaScript
CVE-2012-0754	Flash (Corrupted MP4 Loading)	ActionScript
CVE-2013-0640	API Overflow	JavaScript
CVE-2013-2729	Overflow	File Embedding (BMP)
CVE-2014-0496	Use-After-Free (*toolButton*)	JavaScript
CVE-2015-3203	API Restriction Bypass	JavaScript
CVE-2016-4203	Invalid Font	File Embedding (TFF)
CVE-2017-16379	Type Confusion (*IsAVIconBundleRec6*)	JavaScript
CVE-2018-4990	Use-After-Free (ROP chains)	JavaScript

Table 1 reports a list of the major Adobe Reader vulnerabilities *that have been exploited in the wild* (either with proofs-of-concept or proper malware) in the last decade, along with a brief description of their type and exploitation strategies. Notably, we did not include variants of the same vulnerability in the Table, and we only focused on the most representative ones. Such a list has been obtained by analyzing exploit databases, media sources, security bulletins, and our own file database retrieved from the `VirusTotal` service [32, 34, 40, 72, 79, 91, 98]. According to what we represented in Table 1, there are three primary ways to perform exploitation:

- **JavaScript-based.** These vulnerabilities are exploited by *exclusively* employing `JavaScript` code, and it is the most common way to perform exploitation. The attack code can be scattered through multiple objects in the file or it can be contained in one single object (especially in older exploits).
- **ActionScript-based.** These vulnerabilities exploit the capabilities of Adobe Reader of parsing Flash (`ActionScript`) files due to the proper integration between Reader and Adobe Flash Player. `ActionScript` code can be used in combination with `JavaScript` to attain more effective exploitation.

- **File Embedding.** This exploitation technique resorts to external file types, such as `.bmp`, `.tiff`, and `.exe`. Typically, the exploitation is triggered when specific libraries of Adobe Reader attempt to parse such files. It is also possible to embed other PDF files: however, this is not considered as an exploitation technique, but more as a way to conceal other attacks (see the next sections for more details).

From this list, it can be observed that, although numerous vulnerabilities are still disclosed on Adobe Reader, only a few have been recently exploited in the wild. Such a tiny number of exploited vulnerabilities reflects the fact that PDF files are now less preferred as exploitation vectors by attackers. However, things can unexpectedly change when new, dangerous vulnerabilities are disclosed (such as `CVE-2018-4990`). In the following, we provide a more detailed description of the previously mentioned exploitation techniques by also providing concrete examples from existing vulnerabilities.

JavaScript-based Attacks. `JavaScript`-based attacks are the most used ones in PDF files due to the massive support that the file format provided to this scripting language. In particular, Adobe introduced specific API calls that are only supposed to be used in PDF files (their specifications are contained in the official Adobe references [2]) and that can be exploited to execute unauthorized code on a machine.

According to the vulnerability types contained in Table 1, multiple vulnerabilities can be exploited through `JavaScript` code. Such vulnerabilities can be organized in multiple categories, which we describe in the following:

- *API-based Overflow.* This vulnerability type typically exploits wrong argument parsing for specific API calls that belong to PDF parsing library, thus allowing attackers to perform buffer overflow or ROP-based attacks[3] to complete the exploit. Typical examples of vulnerable APIs are `util.printf` and `Collab.getIcon`, which were among the first ones to be exploited when PDF attacks started to be massively used.
- *Use-after-free.* This vulnerability type is based on accessing memory areas that have been previously freed (and not re-initialized). Normally, such behavior would make the program crash. However, it could allow arbitrary code execution in a vulnerable program.
- *Malformed Data.* This vulnerability type is triggered by compressed malformed data that get decompressed at runtime. Such data is typically stored in streams.
- *Type Confusion.* This vulnerability type may affect functions that receive void pointer as parameters. Typically, the pointer type is inferred by checking some bytes of the pointed object. However, such bytes can be manipulated to make even an invalid pointer type be recognized as valid, thus allowing arbitrary code execution.

A large number of vulnerabilities and exploitation types allow attackers to carry out malicious actions that are often not easy to detect. The evolution of the employed exploitation techniques becomes particularly evident if we observe the differences between the first API overflows (e.g., `CVE-2008-0655`) and the most recent attack strategies. We expect that this trend will further evolve with more sophisticated techniques.

ActionScript-based Attacks. As PDF files can visualize Flash content through Adobe Reader support to Adobe Flash technologies, one way to exploit Reader is by triggering vulnerabilities of its Flash component by embedding malicious ShockWave Flash (SWF) files and `ActionScript` code. Normally, such code is used in combination with `JavaScript`: while executing

[3]Return Oriented Programming, an exploiting strategy that leverages return instructions to create shellcodes.

`ActionScript` triggers the vulnerability itself, the rest of `JavaScript` code carries out and finalizes the exploitation. This exploitation technique was particularly popular in 2010 and 2011.

In a similar way to what we described for `JavaScript`, multiple vulnerabilities can be exploited by using `ActionScript`. In the following, we describe the prominent ones:

- *Memory Corruption*. This vulnerability occurs when specific pointer values in memory get corrupted in a way that they point to other memory areas controlled by the attacker. It is the most-used way to exploit Flash components.
- *ByteCode Verification*. This vulnerability allows attackers to execute code in uninitialized areas of memory.
- *Corrupted File Loading*. This vulnerability is triggered when parsing specific, corrupted video files.

Generally speaking, vulnerabilities that affect the Flash components of Adobe Reader are more complex to exploit than others. This is because two types of scripting code must be executed, and exploitation is typically carried out in two stages (first, `ActionScript` execution; then, `JavaScript` code). For this reason, attackers typically prefer exploits that are simpler to be carried out. Moreover, Flash-based technologies will be dismissed in some years, giving attackers further reasons not to develop further exploits.

File Embedding. This vulnerability exploits the presence of embedded, malformed content inside the PDF file. Typically, decoding such content leads to automatic memory spraying, which can be further exploited to execute code. The file contents that are mostly used for such attacks are the ones related to images (such as `BMP, TIFF`) or fonts (such as `TFF`). In other cases, such as the direct execution of `EXE` payloads, the content is not necessarily malformed but simply directly executed (this also avoids execution of malicious `JavaScript` code to exploit the vulnerability further). The execution of `EXE` payloads can also lead to the generation of additional, malicious payloads (such as `VBS`), whose final goal is dropping the final piece of malware.

3 MACHINE LEARNING FOR PDF MALWARE DETECTION

Machine learning is nowadays widely used to detect document-based infection vectors. In particular, concerning PDF files, multiple detectors were developed in the past decade that implemented such technology. Therefore, the aim of this section is providing an overview of the characteristics of such detectors. However, as this survey focuses on the implications of adversarial attacks against machine learning systems, this section only concerns systems that employ supervised machine learning to perform detection, meaning that we will not discuss PDF malware detectors that employ rule-based or non-supervised approaches (e.g., References [54, 58, 78, 84, 93, 94, 100, 101]). For a more detailed description of such systems, we refer the reader to more general purpose surveys [31, 71].

The primary goal of machine-learning detectors for malicious documents is discriminating between benign and malicious files. They can operate by analyzing and classifying information retrieved either from the structure or the content of the document. More specifically, all systems aimed to detect malicious PDF files share the same basic structure (reported in Figure 2), which is composed of three main components [62]:

(1) **Pre-processing.** This component parses PDF files by isolating the data that are crucial for detection. For example, it can extract `JavaScript` or `ActionScript` code, select specific keywords or metadata, and so forth.
(2) **Feature Extraction.** This component operates on the information extracted during the pre-processing phase by converting it to a vector of numbers. Such a vector can represent,

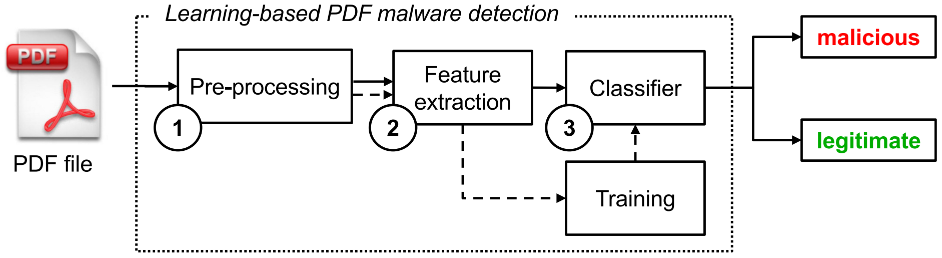

Fig. 2. Graphical architecture of a learning-based PDF malware detection tool.

Table 2. An Overview of the Main Characteristics of Current PDF Malware Detectors

Detector	*Tool*	*Year*	Pre-processing		Features	Classifier
Shafiq et al. [80]	N-Gram	2008	Static	Custom	RAW Bytes	Markov
Tabish et al. [89]	N-Gram	2009	Static	Custom	RAW Bytes	Decision Trees
Cova et al. [23]	Wepawet	2010	Dynamic	JSand	JS-based	Bayesian
Laskov and Šrndić [53]	PJScan	2011	Static	Poppler	JS-based	SVM
Maiorca et al. [64]	Slayer	2012	Static	PDFID	Structural	Random Forest
Smutz and Stavrou [82]	PDFRate-v1	2012	Static	Custom	Structural	Random Forest
Šrndić and Laskov [85, 95]	Hidost	2013	Static	Poppler	Structural	Random Forest
Corona et al. [22]	Lux0R	2014	Dynamic	PhoneyPDF	JS-based	Random Forest
Maiorca et al. [61]	Slayer NEO	2015	Static	PeePDF+Origami	Structural	Adaboost
Smutz and Stavrou [83]	PDFRate-v2	2016	Static	Custom	Structural	Classifier Ensemble

for example, the presence of specific keywords or API calls, or also the occurrences of certain elements in the file.

(3) **Classifier.** The proper learning algorithm. Its parameters are first tuned during the training phase to reduce overfitting and guarantee the highest flexibility against polymorphic variants.

According to the three components described above, Table 2 provides a general overview of machine learning–based PDF detectors that have been released since 2008.

Note how each system employs a combination of different component types (e.g., a specific pre-processing module with a specific learning algorithm or feature extractor). Therefore, we structure the remainder of the section by describing how each component can be characterized, along with its strengths and weaknesses (which will be crucial when such components are observed from the perspective of adversarial machine learning).

3.1 Pre-processing

As reported in the previous sections, the pre-processing phase is crucial to select data that are used for detection. Table 2 showed that there are two major types of pre-processing: *static* and *dynamic*. Static pre-processing analyzes the PDF file without executing it (or its contents). Dynamic pre-processing employs techniques such as sandboxing or instrumentation to execute either the PDF file or its `JavaScript` content. While static analysis is considerably faster and does not require many computational resources, dynamic analysis is much more effective at detecting obfuscated samples, meaning that certain obfuscation routines are almost impossible to be de-obfuscated automatically (in many cases there are multiple stages of obfuscation). As employing a sandboxed

Table 3. An Overview of Third-party Parsers Employed by Machine Learning–based PDF Malware Detectors. Each parser has been evaluated concerning three key components of the file, according to three different degrees of complexity.

Parser	PDF Structure	JavaScript	Embedded Files
Origami [52]	Complete	Partial (Code Analysis)	Complete
JSand [23]	None	Complete	Partial (Analysis)
PDFId [86]	Partial (Key Analysis)	None	None
PeePDF [33]	Partial (Obj. Analysis)	Partial (Code Analysis)	Complete
PhoneyPDF [90]	None	Complete	None
Poppler [36]	Complete	Partial (Extraction)	Complete

Adobe Reader is quite expensive computationally and not practical for the end-user, most detection systems rely on static parsing.

When describing pre-processing tools, we generally divide them into two categories, which we list in the following:

- **Third-party Processing.** This category includes parsers that have been already developed and tested and that do not directly depend on detection tools, thus being included as external modules in the system. The main advantage of employing third-party parsers is that they include many functionalities that are less prone (although not immune, as we will discuss more in detail later) to bugs. However, they can also embed unneeded functionalities and can be quite heavy computationally.
- **Custom Processing.** This category includes parsers that have been written from scratch, to adapt to the information required from the detection system. As such, parsers are tailored to the operations of the detectors, their functionalities are rather limited and much prone to bugs, as typically they have not been extensively tested.

Table 2 clearly shows that the favorite developers' choice is relying on already-existent tools, mostly because of their resilience to bugs and capabilities to adapt to most file variants. For this reason, in the following, we provide a more extensive description of third-party parsers, which is summarized in Table 3.

The table is organized as follows: Each parser analyzes three main elements of the PDF file: the *PDF structure*, the embedded `JavaScript` *code*, and the presence of *embedded files* of any type (including further PDF files). Such elements are analyzed with three degrees of complexity: When the parser can completely analyze these elements, we use the term *Complete*; when only certain operations (reported in brackets in the Table) can be performed, we employ the term *Partial*; finally, when no support is provided, we use the term *None*. The impact of such complexity degrees depends on the analyzed PDF element. In the following, we provide a more detailed description of each degree of complexity for each PDF element:

- **PDF Structure.** It refers to all elements of the PDF structure that are not related to embedded code, such as direct or indirect objects, metadata, and so on. When parsers completely support PDF Structure, it means that they can not only *extract* and analyze object and metadata, but also that they can perform *structural changes* to the file, such as object injection or metadata manipulation. When the support is partial, we typically refer to parsers that are only able to analyze objects, but not to manipulate them. When the support is set to None, it means that the PDF Structure is not analyzed at all. `Poppler` [36] and `Origami` are the only parsers that provide the possibility of properly injecting and manipulating content inside the file.

- **JavaScript.** It refers to the embedded JavaScript code inside the file. When the support to JavaScript is complete, it means that the code can be either statically and dynamically analyzed (to overcome de-obfuscation), for example through instrumentation. When the support is partial, it means that the code can be only statically analyzed (or even only extracted, as it happens in Poppler [36]), leading to some limitations when heavy obfuscation is employed. Finally, when no support is provided, the JavaScript code is not even extracted. JSand and PhoneyPDF [23, 90] are the only parsers that completely support JavaScript instrumentation and execution.
- **Embedded Files.** It refers to the capability of parsers to extract or inject embedded files (such as executable, office documents, or even other PDF files). When the support is complete, parsers can either extract or inject embedded files into the original PDF. When the support is partial, embedded files can only be extracted (or analyzed). Finally, when no support is provided, it means that embedded files cannot be extracted. Origami, PeePDF, and Poppler [33, 36, 52] support extraction and analysis of embedded contents.

From the description provided by Table 3, it can be inferred that no parsers can extract or manipulate all elements of the PDF file, although some of them allow for more functionalities. For this reason, the choice of the parser to be used is related to the type of information that is needed by the learning algorithm to perform its functionality. In the following, we provide a brief description of each third-party parser.

- **Origami** [52]. This parser, entirely written in Ruby, allows users to navigate the object structure of PDF files, to craft malicious files by injecting code or other objects, to decompress and decrypt streams, and so forth. Moreover, it embeds popular information to recognize JavaScript API-based vulnerabilities (see Section 2.2).
- **JSand** [23]. This parser was part of the Wepawet engine to perform dynamic analysis of PDF files. It could execute the embedded JavaScript code to extract API calls and de-obfuscate code. Moreover, it could inspect embedded executables to reveal the presence of additional attacks. Unfortunately, the Wepawet service is currently not available; hence, it is not possible to test JSand anymore.
- **PDFId** [86]. This parser has been developed to extract the PDF *name objects* (see Section 2). It does not perform additional analysis on embedded code or files.
- **PeePDF** [33]. This parser, entirely written in Python, can perform a complete analysis of the PDF file structure (without being able to inject objects). It allows to inject and extract embedded files, and it provides a basic static analysis of JavaScript code.
- **PhoneyPDF** [90]. This parser, entirely written in Python, performs dynamic analysis of embedded JavaScript code through instrumentation. More specifically, it emulates the execution of the JavaScript code embedded in a PDF file to extract API calls that are strictly related to the PDF execution. This parser does not perform any structural analysis or embedding files extraction.
- **Poppler** [36]. Poppler is a C++ library that is used by popular, open-source software such as X-Pdf to render the contents of the file. For this reason, the library features complete support to PDF Structure parsing and managing, as well as JavaScript code extraction and injection of embedded files.

Concerning *custom parsers*, it is important to observe that we could not access the tools that adopted such parsers, as their source was not publicly released. Hence, we could only refer to what has been stated in the released papers [80, 82, 83, 89]. While raw bytes parsers [80, 89] only focused on extracting byte sequences from the PDF, the parser adopted by PDFRate [82, 83] analyzed and

Table 4. An Overview of the Feature Types Selected by Each Machine Learning–based PDF Detector

Detector	*Tool*	*Year*	Structural	JS-Based	RAW Bytes
Shafiq et al. [80]	N-Gram	2008	x	x	N-Grams
Tabish et al. [89]	N-Gram	2009	x	x	N-Grams
Cova et al. [23]	Wepawet	2010	x	Execution-Based	x
Laskov and Šrndić [53]	PJScan	2011	x	Lexical	x
Maiorca et al. [64]	Slayer	2012	Keywords	x	x
Smutz and Stavrou [82]	PDFRate-v1	2012	Metadata	x	x
Šrndić and Laskov [85, 95]	Hidost	2013	Key. Sequence	x	x
Corona et al. [22]	Lux0R	2014	x	API-Based	x
Maiorca et al. [60, 61]	Slayer NEO	2015	Keywords	API-Based	x
Smutz and Stavrou [83]	PDFRate-v2	2016	Metadata	x	x

Each field of the table further specifies the type of feature employed by each detector. Worth noting, when a specific feature type is not used by the detector, we put an x on that field.

extracted the object structure of the PDF file, with a particular focus on PDF metadata. However, the latter parser has been used as a test-bench for adversarial attacks, and researchers proved it could be easily evaded References [19, 96] (see the next sections).

3.2 Feature Extraction

Feature extraction is essential to PDF malware classification. In this phase, data obtained from the pre-processing phase are further parsed and transformed into vectors of numbers of a fixed size. Table 4 provides an overview of the feature types that are used by each PDF detector. We can divide the employed feature types into three categories:

- **Structural.** These features are related to the PDF structure, and most concern the presence or the occurrence of specific keywords (name objects) in the file. Others include metadata or the presence/count of indirect objects or streams.
- **JS-Based.** These features are related to the structure of `JavaScript` code. Most of them concern lexical characteristics of the code (for example, the number of specific operators in the file), used API calls, or information obtained from the code behavior (e.g., when shell-codes are embedded in the attack).
- **Raw Bytes.** This category includes features that concern sequences of bytes taken as n-grams (i.e., groups of n-bytes, where n is typically a small integer).

Most PDF detectors only implement feature-extraction methodologies that include only one type of feature. Byte-level features were among the first ones employed to solve the problem of PDF malware detection, and the very first works in the field mostly adopted them [80, 89]. Typically, these features are represented by simple sequences of bytes taken in groups of n, where n is a very small integer. The reason for such a small number is because the feature space can explode quite easily. Using 1-gram means a total of 256 features, while a 2-gram means 65,536 features. For this reason, this solution has not been considered very practical on standard machine learning models. Moreover, byte-level features do not typically convey explainable information on the file characteristics.

`JavaScript`-based features have been mostly adopted by `Wepawet`,`PJScan`, and `Lux0R` [22, 23, 53]. The general idea behind using these features is to isolate functions that perform dangerous operations, as well as detecting the presence of the obfuscated code that is typically associated with malicious behaviors. `Wepawet` [23] extracted information obtained from the code execution in an emulated environment. Such information is mostly related to code behavior, such as the type of parameters that are passed to specific methods, the sequences of suspicious API calls during execution, the number of bytes that are allocated for string operation (which may be a hint of heap spraying), and so forth. `PJScan` resorted to lexical information extracted from the `JavaScript` code itself, such as the count of specific operators (such as `+` or `()`) that are known for being abused when obfuscated code. Moreover, it performs additional checks on the length of strings to point out the presence of suspicious exploiting techniques (such as buffer overflow or heap spraying). Finally, `Lux0R` exclusively operates on `JavaScript` API calls that belong to the Adobe Reader specifications. In particular, each call is evaluated only once (after being extracted during the pre-processing phase), leading to a binary feature vector of calls. While such features are excellent to analyze and detect attacks that carry `JavaScript` code, they cannot represent other types of attacks, such as the ones that involve the use of embedded files.

Structural approaches have been considerably used in recent years. The main idea, in this case, was trying to address all possible attacks reported in Section 2.2 by using the most general approach possible. This idea also revolves around the concept that malware samples are structurally different from benign ones. For example, they typically feature fewer pages than benign samples, and the representation of the content is significantly scarcer. The first approaches (adopted by `Slayer` and by its extension `Slayer NEO`—which also employed a very reduced number of JavaScript-based features related to known vulnerable APIs) [60, 61, 64] focused on counting the occurrences of specific keywords in the file. The keywords were regarded as relevant when they appeared at least once on enough files in the training set. `Hidost` [85, 95] evolved these approaches by employing sequences of keywords. More specifically, each feature was extracted by walking the PDF tree and evaluating how keywords were sequentially referred. The number of features was limited to 1K, as using all the possible features would have led to an explosion of the algorithm in terms of complexity. Finally, `PDFRate` [82, 83] focused on more general-purpose features that included, among the others, the number of indirect objects, the properties of the stream contents (e.g., the number of upper-case and lower-case characters), and so forth. Moreover, the approach also makes use of information obtained from metadata to identify suspicious behaviors (for example, when popular tools do not generate the PDF file). Albeit structural approaches proved to be effective at detecting even `ActionScript`-based attacks, they exhibit some limitations that will be better described in the next sections.

3.3 Learning and Classification

Feature extraction can be regarded as the process of mapping an input PDF file $\boldsymbol{z} \in \mathcal{Z}$, $\mathcal{Z}$ being the abstract space containing all PDF files, onto a vector space $\mathcal{X} \subseteq \mathbb{R}^{\mathsf{d}}$. In the feature space $\mathcal{X}$, each file is represented as a numerical vector $\boldsymbol{x}$ of d elements. From a mathematical perspective, we can thus characterize the feature-extraction process as a mapping function $\phi : \mathcal{Z} \mapsto \mathcal{X}$ that maps PDF files onto a vector space. Thanks to this abstraction, it is possible to use any learning algorithm to perform classification of PDF documents. All PDF malware detectors resort to *supervised approaches*, i.e., they require the labels of the training samples to be known. More specifically, a learning algorithm is *trained* to recognize a set of known training samples $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^{\mathsf{n}}$, labeled as either legitimate ($y = -1$) or malicious ($y = +1$).[4] During this process, the parameters of the learning algorithm (if any) are typically set according to some given performance requirements. After training, the learning algorithm provides a classification function $f : \mathcal{X} \mapsto \mathbb{R}$ that can be used to assign a real-valued score to each input sample $\boldsymbol{x}$ at test time. Without loss of generality, we can assume here that $\boldsymbol{x}$ is classified as malicious (positive) if $f(\boldsymbol{x}) \geq 0$, and legitimate

[4]Without loss of generality, we assume here that the malicious (legitimate) class is assigned the positive (negative) label.

Table 5. Results Attained at PDF Malware Detection by the Most Popular Machine Learning–based Detectors

Detector	*Tool*	*Year*	Mal. Samples	Ben. Samples	Train. Perc. (%)	TP@FP (%)	F1
Laskov and Šrndić [53]	PJScan	2011	15,279	960	50	71.94@16.35	0.832
Maiorca et al. [64]	Slayer	2012	11,157	9,989	57	99.5@0.02	0.989
Smutz and Stavrou [82]	PDFRate-v1	2012	5,297	104,793	9.1	93.27@0.02	0.801
Šrndić and Laskov [85, 95]	Hidost	2013	82,142	576,621	33.7	99.73@0.06	0.825
Corona et al. [22]	Lux0R	2014	12,548	5,234	70	99.27@0.05	0.986
Maiorca et al. [60, 61]	Slayer NEO	2015	11,138	9,890	57	99.81@0.07	0.969
Smutz and Stavrou [83]	PDFRate-v2	2016	5,297	104,793	9.1	99.5@0.05	0.667

The table reports the overall malicious and benign samples, the training set percentages, the True Positives (TP) at False Positives (FP) rate, and the overall F1-score.

(negative) otherwise. Notably, the appropriate learning algorithm is selected depending on the features that are used for classification and on the training data at hand. Decision trees have been used by most PDF malware detectors and proved to be very effective to address this problem [22, 61, 64, 82, 83, 89, 95]. In particular, ensemble models such as Random Forests or Boosting showed very high accuracy under limited false positives. The underlying reason is that such classifiers adapt well to heterogeneous information, and in particular to discrete feature values such as counts. However, depending on the feature types, other solutions may be adopted. For example, `PJScan` [53] resorts to Support Vector Machines (SVMs), `Wepawet` [23] to Bayesian classifiers, and Shafiq et al. [80] to Markov models (to deal with *n*-gram-based feature representations). Nevertheless, tree-based classifiers generally reported better performances at detecting PDF malware (see Section 3.4).

3.4 Detection Results

In the following, Table 5 presents the results attained on PDF malware detection by the most popular machine learning–based systems in the wild. Notably, the goal of the table is *not* to show which system performs best but to provide indications on how such detectors generally cope with PDF attacks in the wild. Direct comparison among the performances attained by all systems would not be fair, considering that each system was trained and tested on different samples, with different training/test splits and classifiers. Therefore, to build a table that is coherent and meaningful, we followed these guidelines:

- We considered the results attained by systems that exclusively detected PDF malware. Therefore, we ruled out References [23, 80, 89], as their datasets also included other malware types besides PDF.
- Our table includes the overall number of malicious and benign samples, the percentage of the dataset used for training the system, the True Positive (TP) rate attained in correspondence of the relative False Positive (FP) rate (TP@FP), and the relative F1-score. TP@FP is among the most-used performance measure in malware analysis, and it is very useful to indicate the performances of the systems at specific FP values. Keeping a low FP rate guarantees proper system usability (too many false alarms would disrupt the overall user experience). Note that we referred to values that were *explicitly* stated in each *original* work (we did not consider any cross-analysis).
- Notably, many papers adopted significantly imbalanced datasets in their analysis. For this reason, we used F1 as an overall measure that considers the distribution of the data as a crucial element to measure performance. We point out that systems with higher F1-score

are not necessarily better than the others. We included this evaluation to provide the reader with another perspective from which to evaluate the results.

Notably, these choices have been driven by the heterogeneous nature of the examined works. In particular, many of them did not report precise numbers of employed benign and malicious test samples, but only the overall test-set percentage concerning a precise number of malicious and benign samples. For this reason, when calculating the F1-scores for each system, we assumed that the test percentages were equally applied for benign and malicious samples.

Some of the examined works reported multiple results attained by changing the features, classification parameters, and data distribution [53, 82, 85, 95]. Hence, for each system, we focused on the following information (the reader may check the original works for more details):

- **PJScan** [53]. We considered as malicious the files that were regarded as *detected* in the original paper, and as benign the files that were regarded as *undetected*. We reported the performances obtained with native tokens and on `JavaScript` files only, as `PJScan` does not make any analysis of non-`JavaScript` files.
- **PDFRate-v1** [82]. We reported the performances related to the lowest number of false positives that are stated in the original work.
- **Hidost** [85, 95]. We reported the result attained by the work in Reference [95], as it clearly states the attained TP and FP percentages.
- **PDFRate-v2** [83]. As performances were stated in the paper by considering multiple thresholds for classification, we report the results attained at 0.5 threshold. The choice was made to obtain false positive values that were similar to the ones attained by the other systems.

The attained results show some interesting trends. First, almost all systems attained very high accuracy at detecting PDF malware with low false positives. Such positive results mean that various information (feature) types can be equally effective at detecting PDF malware.

Second, there is a consistent F1-score difference between `Slayer` [64] and its evolved `Slayer NEO` [61] version, and between `PDFRate-v1` [82] and `PDFRate-v2` [83], where the attained F1-scores decreased in the most recent versions of the tools. In particular, we observe that this decrease is due to a higher number of false positives. Notably, this effect is particularly evident on `PDFRate-v2`, where imbalanced datasets significantly penalize the F1-score when false positives increase. Finally, we observe that other works used most of the dataset for training [22, 60, 61, 64], which means that such systems would require more training data to perform classification correctly.

4 ADVERSARIAL ATTACKS AGAINST PDF MALWARE DETECTORS

In this section, we start by formalizing a threat model for PDF malware detection, inspired from previous work in the area of adversarial machine learning, and then we will use it to categorize existing and potentially new adversarial attacks against such systems.

As highlighted in a recent survey [16], the first attempts to formalize adversarial attacks against learning algorithms date back to the decade 2004–2014 [6, 7, 9, 12, 13, 15, 25, 38, 44, 48, 57], prior to the recent discovery of *adversarial examples* against deep neural networks [39, 88]. PDF malware has provided one of the major case studies in the literature of adversarial machine learning over these years, as its inherent structure allows for fine-grained modifications that adapt well to how adversarial attacks are typically performed. As we will discuss in the remainder of this article, this is true in particular for evasion attacks, i.e., attacks in which PDF malware is manipulated at test time to evade detection. Preliminary attempts in crafting evasive attacks against PDF malware detectors were first described by Smutz and Stavrou [82], and subsequently by Šrndić and Laskov [95], even though they were based on heuristic strategies that were able to mislead linear classification algorithms successfully. To our knowledge, the very first work proposing optimization-based evasion attacks against PDF malware detectors is the work by Biggio et al. [8]. In that work, the authors were able to show that even nonlinear models were vulnerable to *adversarial* PDF malware, conversely to what envisioned in Reference [95]. The attack was done by selecting the feasible manipulations to be performed on each malware sample via a gradient-based optimization procedure, in which the attacker aimed to minimize the classifier's prediction confidence on the malicious class (i.e., to get maximum-confidence predictions on the opposite class, namely, the benign one). Worth remarking, the gradient-based procedure described in that paper has been the first to demonstrate the vulnerability of learning algorithms to optimization-based evasion attacks, even before the discovery of adversarial examples against deep networks [39, 88].

4.1 Threat Modeling and Categorization of Adversarial Attacks

The threat model proposed in this section aims to provide a unified treatment of both attacks developed in the area of adversarial machine learning and attacks developed specifically against PDF malware detectors. As we will see, this does not only enable us to categorize existing attacks under a consistent framework, clarifying the (sometimes implicit) assumptions behind each of them. The proposed threat model will also help us to identify new potential attacks that may threaten PDF malware detectors in the near future, as well as novel research directions for improving the security of such systems, inspired by recent findings in the area of adversarial machine learning.

Leveraging the taxonomy of adversarial attacks initially provided by Barreno et al. [6, 7] and subsequently expanded by Huang et al. [44], Biggio and Roli [16] have recently provided a unified threat model that categorizes adversarial attacks based on defining the *attacker's goal*, her *knowledge* of the target system, and her *capability* of manipulating the input data.[5] In the following, we discuss how to adapt this threat model to the specific case of PDF malware detectors.

4.1.1 Attacker's Goal. The attacker's goal is defined based on the desired security violation. When speaking about systems' security, an attacker can cause **integrity**, **availability,** or even **privacy** violations. Violating system integrity means having malware samples undetected without compromising normal system operation for legitimate users. An example of integrity violation is when benign PDF files are injected with a payload that is undetected by anti-malware engines. In this way, the user still visualizes the file content, but other operations still occur in the background. Availability is compromised when many benign (but also malware) samples are misclassified, effectively causing a denial of service for legitimate users. This situation may occur on PDF files when the attacker injects multiple scripting (benign) instructions to trigger multiple fake alerts due to the presence of codes (which cannot be, alone, indicators of maliciousness). Finally, privacy is violated if the system may leak confidential information about its users, the classification model used, and even the training data used to learn it. For example, an attacker can incrementally change some characteristics of the PDF file (for example, by adding text, using fonts, adding scripting codes, and so forth) and send them to the target classifier to see how it reacts to such changes.

4.1.2 Attacker's Knowledge. The attacker can have different levels of knowledge of the targeted system, including: (*i*) the training data $\mathcal{D}$, consisting of n training samples and their labels, i.e., $(\boldsymbol{x}_i, y_i)^n$; (*ii*) the feature set $\mathcal{X} \subseteq \mathbb{R}^d$, which is strongly related to the PDF detector components depicted in Figure 2 and Table 2 (see Section 3). More specifically, the attacker may know which pre-processing and feature extraction algorithms are employed, along with the extracted

[5]We refer to the attacker as feminine here due to the popular role impersonated by Eve (or Carol) in cryptography.

Table 6. An Overview of the Knowledge Levels Held by an Attacker, According to the Elements of the Knowledge Space

Knowledge Level	$\mathcal{D}$	$\mathcal{X}$	f
White box (PK)	✓	✓	✓
Gray box (LK)	x	✓	x
Black box (ZK)	x	x	x

When a component of the space is fully known, we represent it with a checkmark (✓), while we use an x when the element is partially known or unknown.

feature types; (*iii*) the classification function $f : \mathcal{X} \mapsto \mathbb{R}$ (to be compared against zero for classification), along with the objective function minimized during training (if any), its hyperparameters and, potentially, even the classifier parameters learned after training. The attacker's knowledge can be conceptually represented in an abstract space Θ, whose elements correspond to the aforementioned components (*i*)–(*iii*) as $\boldsymbol{\theta} = (\mathcal{D}, \mathcal{X}, f)$. Depending on the assumptions made on (*i*)–(*iii*), there may be different attack scenarios, described in the following, and compactly summarized in Table 6.

Perfect-Knowledge (PK) White-box Attacks. In this scenario, we assume that the attacker knows everything about the target system, i.e., $\boldsymbol{\theta}_{\text{PK}} = (\mathcal{D}, \mathcal{X}, f)$. Even though this may rarely occur in practical settings, this scenario is useful to provide an upper bound on the performance degradation incurred by the system under attack and to understand how far from the worst case more realistic evaluations are.

Limited-knowledge (LK) Gray-box Attacks. This category of attacks, in general, assumes that the attacker knows the feature representation $\mathcal{X}$, but she does not have full knowledge of the training data $\mathcal{D}$ and the classification function f. In particular, it is often assumed that the attacker can collect a surrogate dataset $\hat{\mathcal{D}}$ resembling that used to train the target classifier from a similar source (e.g., a public repository).[6] Regarding the classification function f, the attacker may know the type of learning algorithm used (e.g., the fact that the classifier is a linear SVM) and, ideally, its hyperparameters (e.g., the value of the regularization parameter C used to learn it), although this may not be strictly required. However, the attacker is assumed not to know the exact classifier's trained parameters (e.g., the weights of the linear SVM after training), but she can potentially get feedback from the classifier about its decisions and labels. Under these assumptions, the attacker can estimate the classification function from $\hat{\mathcal{D}}$ by training a *surrogate classifier* $\hat{f}$. We thus denote this attack scenario with $\boldsymbol{\theta}_{\text{LK}} = (\hat{\mathcal{D}}, \mathcal{X}, \hat{f})$. Notably, these attacks may also include the case in which the attacker knows the trained classifier f (white-box attacks), but optimizing the attack samples against the target function f may not be effective. The reason is that, for certain configurations of the classifier's hyperparameters, the objective function in the white-box case may become too noisy and difficult to optimize with gradient-based procedures, due to the presence of many poor local minima or null gradients (a phenomenon known as *gradient obfuscation* [4, 73]). Therefore, it is preferable for the attacker to optimize the attack samples against a surrogate classifier with a smoother objective function and then test them against the target one. This is a common procedure also used to evaluate the *transferability* of attacks [8, 29, 30, 55, 73].

[6]We use here the *hat* symbol to denote limited knowledge of a given component.

Table 7. An Overview of Adversarial Attacks Against Learning Systems, Adapted from [16]

	Attacker's Goal		
Attacker's Capability	*Integrity*	*Availability*	*Privacy*
Test data	Evasion	-	Model Extraction/Stealing Model Inversion Membership Inference
Training data	Poisoning (Integrity) Backdoor	Poisoning (Availability)	-

Zero-knowledge (ZK) Black-box Attacks. The term *zero knowledge* is typically used in literature to indicate the possibility that the attacker can query the system to obtain feedback on the labels and the provided score and optimize the attack samples in a black-box fashion [20, 26, 73, 92, 102]. However, it should be clarified that the attacker still has some minimal knowledge of the system. For example, she knows that the classifier has been designed to perform specific tasks (e.g., detect PDF files), and she has an idea of what kind of transformations should be made on the samples to attempt evasion. Hence, some details of the feature representation are still known (e.g., the employed feature types, whether static or dynamic analysis is performed, etc.). The same considerations can be done about knowledge of the training data; e.g., it is obvious that a system designed to recognize specific PDF malware has been trained with benign and malicious PDF files. Thus, in agreement with Biggio and Roli [16], we characterize this setting as $\boldsymbol{\theta}_{\text{ZK}} = (\hat{\mathcal{D}}, \hat{\mathcal{X}}, \hat{f})$. Note that using surrogate classifiers is not strictly necessary here [20, 26, 92, 102]; however, it is possible to learn a surrogate classifier (potentially on a different feature representation) and check whether the crafted attack samples *transfer* to the target classifier. Feedback from the classifier's decisions on specifically crafted query samples can be also used to refine and update the surrogate model [73].

4.1.3 *Attacker's Capability.*

The attacker's capability of manipulating the input data is defined in terms of the so-called *attack influence*, as well as by some data manipulation constraints. The attack influence defines whether the attacker can only manipulate **test data** (*exploratory* influence), or also the **training data** (*causative* influence). The latter is possible, *e.g.*, if the system is retrained online using data collected during operation, which can be manipulated by the attacker [7, 13, 16, 44]. Depending on whether the attack is staged at *training* or *test* time, different data manipulation constraints can be defined. These define the changes that can be concretely performed by the attacker to evade the system. For example, to evade malware detection at test time, malicious code must be modified without compromising its intrusive functionality. This strategy may be employed against systems based on static code analysis, by injecting instructions or code that will never be executed [8, 28, 41, 97]. For training-time attacks, instead, the constraints typically impose that the attacker can only control a small fraction of the training set [7, 16, 44, 45, 48]. In both cases, as discussed in [16], the attacker's capability can be formalized in terms of mathematical constraints along with the optimization problem defined to craft the attack samples.

4.1.4 *Summary of Attacks.*

We describe here the potential attacks that can be perpetrated against machine-learning algorithms, according to the assumptions made on the attacker's goal and on her capability of manipulating the input data [16]. Table 7 introduces the set of adversarial attacks that have been considered to date. Notably, each of these attacks can be performed according to different levels of knowledge, as described in Section 4.1.2.

Evasion Attacks. In this setting, the attacker attempts to manipulate test samples to have them misclassified as desired [8]. Evasion attacks are also commonly referred to as *adversarial examples*, especially when staged against deep-learning algorithms for image classification [16, 88]. These attacks exploit specific weaknesses of a previously trained model, and they have been widely used to show test-time vulnerabilities of learning-based malware detectors.

Poisoning Attacks. Poisoning attacks target the training stage of the classifier. The attacker intentionally injects wrongly labeled samples into the training set, aiming to decrease the detection capabilities of the classifier. If the attack aims to indiscriminately increase the classification error at test time, causing a denial of service to legitimate system users, it is referred to as a *poisoning availability* attack [6, 7, 10, 14, 16, 44, 45, 65, 68]. Conversely, if the attack is targeted to only have few samples misclassified at test time, then it is named as a *poisoning integrity* attack [6, 7, 10, 14, 16, 44, 45, 48, 49, 68]. We also include *backdoor attacks* in this category, as they also aim to cause specific misclassifications at test time by compromising the training process of the learning algorithm [21, 42, 46, 56]. Their underlying idea is to compromise a model during the training phase or, more generally, at design time (this may include, e.g., also modifications to the architecture of a deep network by addition of specific layers or neurons), with the goal of enforcing the model to classify *backdoored* samples at test time as desired by the attacker. Backdoored samples may include malware files with a specific signature or images with a specific subset of pixels set to given values. Once the model recognizes this specific signature (i.e., the backdoor), it should output the desired classification. A popular example is the stop sign with a yellow sticker attached on it (i.e., the signature used to activate the backdoor), which is misclassified as a speed-limit sign by the backdoored deep network considered in Reference [42]. The overall intuition is that these vulnerable models can then be released to the public, to be used as open-source pre-trained models in other open-source tools or even commercial products, and consequently make the latter also vulnerable to the same backdoor attacks.

Privacy Attacks. Privacy-related attacks aim to steal information about unknown models for which the attacker is given black-box query access. In these attacks, the attacker sends specific test samples against the target model with the aim of obtaining information about: (*i*) the model itself, via *model extraction* (or *model stealing*) attacks [92]; or (*ii*) the data used to train it, via *model inversion* attacks (to reconstruct the training samples) [35] or *membership inference* attacks (to evaluate whether a given sample has been used to train the target model or not) [69, 81].

In the remainder of this manuscript, we provide detailed insights into each attack concerning the problem of PDF malware detection.

4.2 Evasion Attacks Against PDF Malware Detectors

Research work in attacking PDF malware detectors focused mostly on evasion attacks [8, 9, 82, 95]. As highlighted by our previous categorization of adversarial attacks, evasion attacks aim to violate system *integrity* by manipulating the input PDF file structure at *test time*.

In the specific case of attacks against PDF malware detectors, we can identify two main families of evasion attacks, i.e., *optimization-based* [8, 9, 96, 102] and *heuristic-based* [19, 22, 61–63, 82, 83, 95] attacks. Optimization-based evasion attacks perform fine-grained modifications to the input sample to minimize (maximize) the probability that the sample is classified as malicious (benign). Heuristic-based evasion attacks also aim to create evasive samples, but they are not formulated in terms of an optimization problem; rather, they provide reasonable modifications that are expected to cause misclassifications of malware samples at test time, including, e.g., trying to mimic the structure of benign files. Within these two broad categories of attacks, we will then specify whether each attack is carried out under a white-box, gray-box, or black-box attack scenario, to highlight the set of assumptions made on the attacker's knowledge, as encompassed by our threat model. We

Table 8. An Overview of the Evasion Attacks Proposed Thus Far Against PDF Malware Detectors

Work	*Year*	Heur./Opt.	Knowl. (B/G/W)	Target System(s)	Real Sample
Smutz and Stavrou [82]	2012	Heur.	G	PDFRate-v1	No
Šrndić and Laskov [95]	2013	Heur.	B,W	Hidost	No
Biggio et al. [8, 9]	2013	Opt.	G,W	Slayer	No
Maiorca et al. [63]	2013	Heur.	B	Wepawet, PDFRate-v1, PJScan	Yes
Corona et al. [22]	2014	Heur.	B	Lux0R	No
Šrndić and Laskov [96]	2014	Opt.	W	PDFRate	Yes
Maiorca et al. [60, 61]	2015	Heur.	B	Wepawet, PDFRate-v1, PJScan, Slayer N.	Yes
Carmony et al. [19]	2016	Heur.	B	PDFRate-v1, PJScan	Yes
Xu et al. [102]	2016	Opt.	B	PDFRate-v1, Hidost	Yes
Smutz and Stavrou [83]	2016	Heur., Opt.	B,W	PDFRate-v2	Yes
Maiorca and Biggio [62]	2019	Heur.	B	PDFRate-v1, PJScan, Slayer N., Hidost	Yes

This taxonomy considers the attack family (i.e., optimization- or heuristic-based), the attacker's knowledge of the target system (i.e., whether she has white-box, gray-box, or black-box access), the target system(s), and whether the attack has been staged at the input level (i.e., creating the adversarial PDF malware samples) or at the feature level (i.e., only manipulating their feature values without actually creating the PDF files).

will also discuss the practical difficulties associated with concretely staging the attack against a real system, which are inherent to the creation of the evasive (or *adversarial*) PDF malware samples.

This issue has been widely known in the literature of adversarial machine learning, especially for optimization-based attacks, under the name of *inverse feature-mapping problem* [8, 12, 13, 44]. This problem arises from the fact that most of the optimization-based attacks proposed thus far craft evasive instances only in the *feature space*, i.e., by directly manipulating the values of the feature vector $\boldsymbol{x}$, without creating the corresponding evasive PDF malware sample. The modifications are typically constrained to make it possible to create the actual PDF file, at least in principle; however, how to invert the representation $\boldsymbol{x}$ to obtain the corresponding PDF file as $z = \phi^{-1}(\boldsymbol{x})$ has only rarely been considered. This point is particularly of interest and goes beyond the problem of adversarial PDF malware; for example, adversarial attacks manipulating Android and binary malware are subject to the same issues [28, 51]. The problem here amounts to creating a malware sample exhibiting the desired feature vector, modified to evade the target system, which is not always an easy task. For example, in the case of PDF malware, injecting material may change the file *semantics* (i.e., the file may behave differently than the original one) or even break the malicious functionality of the embedded exploitation code.

We summarize the attacks proposed thus far to craft *adversarial PDF malware* in Table 8. Our taxonomy is organized along four axes: (*i*) the family of evasion attacks (i.e., either heuristic-based or optimization-based); (*ii*) the attacker's knowledge of the learning model (i.e., whether she has white-box, gray-box, or black-box access to the target classifier); (*iii*) the target system(s); and (*iv*) whether the attack has been staged at the *input level* (i.e., by creating the adversarial PDF malware sample) or at the *feature level* (i.e., by only modifying the feature values of the attack samples, without creating the corresponding PDF files).

We provide below a more detailed description of the attacks that have been performed against PDF malware detectors, following the proposed categorization (Sections 4.2.1–4.2.2). We then discuss some insights on how to tackle the inverse feature-mapping problem and create adversarial PDF malware samples. As we will see, this can be achieved by injecting content into PDF files through the exploitation of vulnerabilities in their parsing mechanisms (Section 4.2.3).

4.2.1 Optimization-based Evasion Attacks. Evasion attacks (also known as *adversarial examples*) consist of minimizing the classifier's score on the malicious class $f(\boldsymbol{x}')$ with respect to the input adversarial sample $\boldsymbol{x}'$ [8]. The changes to the input vector $\boldsymbol{x}'$ are constrained to reflect feasible manipulations on the source malicious PDF file $\boldsymbol{x} = \phi(\boldsymbol{z})$. In the case of PDF malware detection, they are typically restricted to only consider the injection of content. The problem of optimizing adversarial PDF malware samples has been originally formulated by Biggio et al. [8] as:

$$\arg\min_{\boldsymbol{x}'} \quad f(\boldsymbol{x}') - \lambda g(\boldsymbol{x}'), \tag{1}$$

$$\text{s.t.} \quad \boldsymbol{x} \leq \boldsymbol{x}', \text{ and } \|\boldsymbol{x}' - \boldsymbol{x}\|_1 \leq \varepsilon, \tag{2}$$

where the first constraint $\boldsymbol{x} \leq \boldsymbol{x}'$ reflects content injection (i.e., each component of $\boldsymbol{x}'$ has to be greater or equal to the corresponding one in $\boldsymbol{x}$), and the second one bounds the number of injected elements to be not greater than ε (as it essentially counts in how many elements $\boldsymbol{x}$ and $\boldsymbol{x}'$ are different). The function $g(\boldsymbol{x}')$ reflects how similar the manipulated sample $\boldsymbol{x}'$ is to the benign distribution, and λ is a trade-off parameter. This trick was introduced in Biggio et al. [8, 9] to facilitate evasion by avoiding poor local minima of $f(\boldsymbol{x}')$ (which do not typically allow the attack algorithm to find a correct evasion point). In fact, when λ is small, the attacker may find an evasion point by typically injecting very few objects, and such a point is normally quite different from both the malicious and benign training samples. When λ increases, the attack point is modified in a more significant manner, but it also becomes harder to distinguish its manipulated feature representation to that of benign samples. The aforementioned optimization problem is usually solved by projected gradient descent [8, 9], and it has been used in follow-up work to generate adversarial PDF malware [83, 96], adversarial binaries [51], and adversarial images against deep networks [66]. With respect to the area of adversarial images against deep networks, many different algorithms have been proposed to generate the so-called *adversarial examples* [18, 39, 59, 74, 88]. They are nevertheless all based on the idea of formulating the attack as an optimization problem and solve it using different gradient-based optimization algorithms. We refer the reader to Reference [16] for a comprehensive survey on this topic.

To solve the aforementioned problem with gradient-based optimization, the attacker needs to be able to estimate the gradient of the objective function, which in principle requires white-box access to the target classifier $f(\boldsymbol{x}')$ and ideally also to the training data (to estimate the gradient of $g(\boldsymbol{x}')$). However, it has been shown in Reference [8] that such a gradient can be estimated reliably even if the attacker has only gray-box access to the target system. The underlying idea is to first collect a surrogate training set and query the target system to relabel it (e.g., if the target system is provided as an online service).[7] A surrogate learner $\hat{f}$ can then be trained on such data to approximate the target function f, while an estimate $\hat{g}$ of the benign sample distribution g can be obtained directly from the surrogate data. Attacks can be thus staged against the surrogate learner and then *transferred* to the target system. Within this simple trick, one can craft gradient-based attacks with both white-box and gray-box access to the target learner.[8] In the context of PDF malware, white-box and gray-box gradient-based attacks have been used in References [8, 9, 83, 96]. Black-box attacks have been more recently proposed in Reference [102], based on the idea of minimizing the value of $f(\boldsymbol{x}')$ by only querying the target classifier in an iterative manner through using a genetic algorithm. These three families of attack are described below. We also discuss how they can be seen as specific instances of the aforementioned optimization problem.

[7]However, it has been shown that if the surrogate data is well representative of the training data used by the target system, the relabeling procedure is not even required [8, 29].
[8]Despite the fact that follow-up work has defined these attacks as black-box transfer attacks [73], we prefer to name them gray-box attacks, as such attacks implicitly assume that the attacker knows at least the feature representation.

White-box Gradient-based Evasion Attacks. This category includes the gradient-based attacks proposed by Biggio et al. [8, 9] and by Šrndić et al. [96], assuming white-box access to the target model (which includes knowledge of its internal, trained parameters). Biggio et al. [8, 9] focused on evading `Slayer` [64] and considering three different learning algorithms: SVMs with the linear and the RBF kernel, and neural networks. Their results showed that, even with a minimal number of injected keywords (less than 10 against the linear SVM), it is possible to evade PDF malware detection. These results showed, for the first time, that non-linear malware detectors can be vulnerable to test-time attacks (conversely to what had been previously reported in Reference [95]).Although the constrained optimization problem in Eqs. (1)–(2) allows in principe the creation of the corresponding PDF malware samples, this was not concretely demonstrated in Reference [8] (as the analysis was only limited to the manipulation of numerical feature values).

To overcome this limitation, Šrndić and Laskov [96] expanded the work in Reference [8] by performing a practical evasion of `PDFRate-v1` [82]. They considered the same optimization problem with white-box access to the target system but injected the selected features for evasion directly into the malicious PDF file, after the `EOF` tag. Albeit effective at evading the target system, such an injection strategy may be easily countered by improving the parsing process [96]. Notably, as `PDFRate-v1` employs standard decision trees (which are non-differentiable), the authors replaced the classifier with a surrogate, differentiable SVM. Even with this characteristic, the attack can be considered as white-box, as the attacker possesses complete knowledge of the target system. More recently, Smutz et al. [83] tested the same attack against an evolved version of `PDFRate` (which we refer to as `PDFRate-v2`) that adopted a customized ensemble of trees as a classifier. However, due to the non-differentiable nature of the trees, the authors directly replaced them with an ensemble of SVMs. Notably, the attack was performed directly against the SVM ensemble, which was not used as a surrogate. The attained results showed that gradient-based attacks completely evaded the system.

Gray-box Gradient-based Evasion Attacks. This scenario has been explored by Biggio et al. [8, 9] and by Šrndić and Laskov [96]. We point out here its importance, as it shows what happens when gradient-descent-based attacks are performed under more realistic scenarios. Notably, it would be unfeasible for the attacker to have perfect knowledge of the target system (including the classifier's trained parameters). For this reason, as it is easily predictable, the efficacy of gradient-descent attacks is reduced by the fact that the attacker has to learn a surrogate algorithm (and to use surrogate data) to mount the attack. However, experiments performed against `Slayer` showed that all classifiers were completely evaded by simply increasing the number of changes to the file. Most were evaded after 30–40 changes, which means injecting that number of keywords. This attack showed that, even under realistic scenarios, gradient descent can be a very effective approach to evade PDF malware detectors.

Black-box GA-based Evasion Attacks. This attack has been proposed by Xu et al. [102]. The underlying idea is to optimize the objective function in Equation (1) (with $\lambda = 0$, i.e., to only minimize $f(\boldsymbol{x}')$) using a genetic algorithm (GA) that iteratively queries the target system in a black-box scenario. In this case, the input PDF malware sample is directly modified by injecting and deleting features (accordingly, the constraint $\boldsymbol{x} \leq \boldsymbol{x}'$ in Equation (2) is not considered here), and then dynamic analysis is used to verify that its malicious functionality has been preserved. This attack has been staged against `PDFRate-v1` and `Hidost` and proven to be very effective at evading both systems. While this attack is the only one that considers removal of objects from PDF files, it is also very computationally demanding. The underlying reason is twofold: (*i*) the genetic algorithm used in the optimization process requires performing a large number of queries (as it does not exploit any structured information like gradients to drive the search in the optimization space), and (*ii*) the attack requires running dynamic analysis on the manipulated samples. While this may not be an issue for an attacker whose goal is only developing successful attacks, GA-based evasion may not be suitable for generating attacks under limited time, computational resources and, most importantly, number of queries to the target system.

4.2.2 Heuristic-based Evasion Attacks. Heuristic-based evasion attacks had been largely used before optimization-based attacks were found to be more effective. They include strategies that do not directly optimize an objective function, and this is why they are typically less effective than optimization-based attacks. Heuristic-based attacks attempt evasion by making malicious samples more similar to benign ones. This effect is typically obtained either by adding benign features to malicious files (*mimicry*) or by injecting malicious payload in benign files (*reverse mimicry*). According to the knowledge possessed by the attacker, we distinguish among the three evasion strategies discussed below.

White-box Mimicry. The main idea of this attack is manipulating malware samples to make them as close as possible to the benign data (in terms of their conditional probability distribution or distance in feature space), under white-box access to the trained model, i.e., by injecting the most discriminant features for the target classifier directly into the input sample. This implementation was adopted in Reference [82] to bypass `PDFRate-v1`. The attained results showed that it was possible to decrease the classifier accuracy by more than 20% by only injecting the top six discriminant features in each malware sample. A similar approach was adopted in Reference [95], where the authors tested the robustness of `Hidost` by injecting the features that would influence the classifier decision the most. The attained result showed that it was possible to evade the tree-based classifiers with high probability, but this was not the case for nonlinear SVMs with the RBF kernel.

Black-box Mimicry. In this strategy, the attacker injects into a malicious sample all the objects contained in a benign file, aiming to subvert the classifier decision. To this end, the attacker only needs to have black-box access to the target classifier and to collect some benign PDF files. The easiest way to perform this attack is to blindly copy the entire content of a benign PDF file into the malicious sample. This approach was adopted by Corona et al. [22] to test the robustness of `Lux0R`. In particular, the authors added all the JavaScript API-based features belonging to multiple benign samples to the given malicious PDF files and measured how the classifier detection was affected. They also measured the changes in detection rate as the number of injected files increased. Results showed that, due to the dynamic nature of the features employed, the classifier was still able to detect most of the malicious samples in the wild, despite the modifications they received. Šrndić and Laskov [95] also adopted a similar approach to verify the robustness of `Hidost` (trained, in this case, with an RBF SVM—a different case from the tests they made with white-box mimicry against tree classifiers), showing that their approach was robust against this kind of mimicry.

Black-box Reverse Mimicry. In this attack, the attacker injects a specifically crafted, malicious payload into a file that is recognized as benign by the classifier by only exploiting black-box access to the classifier. The idea is that the corresponding feature modifications should not be sufficient to change the classification of the source sample from benign to malicious [61–63, 83]. Such a strategy is exactly the opposite of mimicry attacks: while the latter inject benign information into a malicious file, reverse mimicry tends to minimize the amount of malicious information injected (by changing the injected payload when the attack fails). This difference is depicted in the flow diagrams of Figure 3.

There are three types of reverse-mimicry attacks, according to different types of malicious information that can be injected:

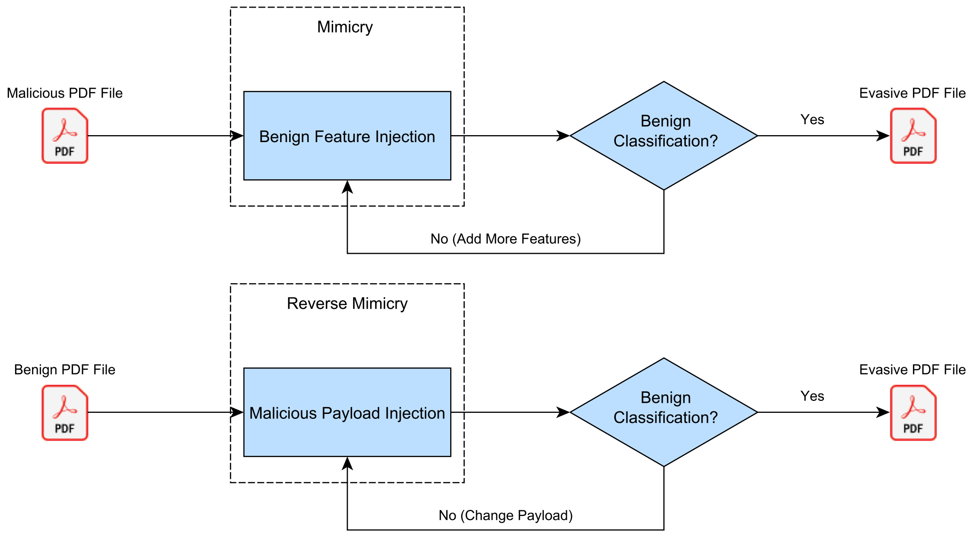

Fig. 3. Conceptual flow of *mimicry* (top) and *reverse-mimicry* (bottom) attacks.

- *JS Embedding*. This approach injects `JavaScript` codes directly inside the file objects. Current implementations feature the injection of only one `JavaScript` code, but it is technically possible to scatter the code through multiple objects in the file. Notably, this would increase the probability of the attack to be detected, as more keywords have to be used to indicate the presence of the embedded code.
- *EXE Embedding*. This technique exploits the `CVE-2010-1240` vulnerability, thanks to which it is possible to run an executable from the PDF file itself directly. In the implementation proposed by Maiorca et al. [63], the EXE payload was injected through the creation of a different version.
- *PDF Embedding*. This strategy features the injection of malicious PDF files in the objects of the target PDF file. More specifically, attackers can inject specific keywords that cause the embedded file to open automatically. This technique can be particularly effective, especially because multiple embedding layers can be easily created (for example, embedding a PDF file in another PDF file, which is finally embedded in another file). In Reference [61], `PeePDF` was employed to carry out such embedding strategy. However, the embedding process is prone to bugs, due to the complexity of the PDF format. Nevertheless, it is possible to employ libraries such as `Poppler` to improve the injection process.

The efficacy of reverse-mimicry attacks has been explored in various works. Maiorca et al. [63] demonstrated that all reverse-mimicry variants were able to evade systems that adopted structural features, such as `PDFRate-v1`. Further works [60, 61, 83] proposed possible strategies to mitigate such attacks, which will be described in moredetail in Section 5.

We now summarize the results attained by testing evasive examples created with reverse-mimicry strategies. In particular, we report the results attained by Reference [62], in which 500 samples for each reverse-mimicry variant (for a total of 1,500 samples) were created and tested against multiple systems in the wild [53, 60, 82, 83, 95]. Such comparison is the most recent and fair between multiple systems on evasive datasets. All systems (except for `PDFRate-v1`, which was provided as an online service[9]) were trained with the same dataset composed by more than 20K

[9]`PDFRate-v2` was not available during the experiments in Reference [62].

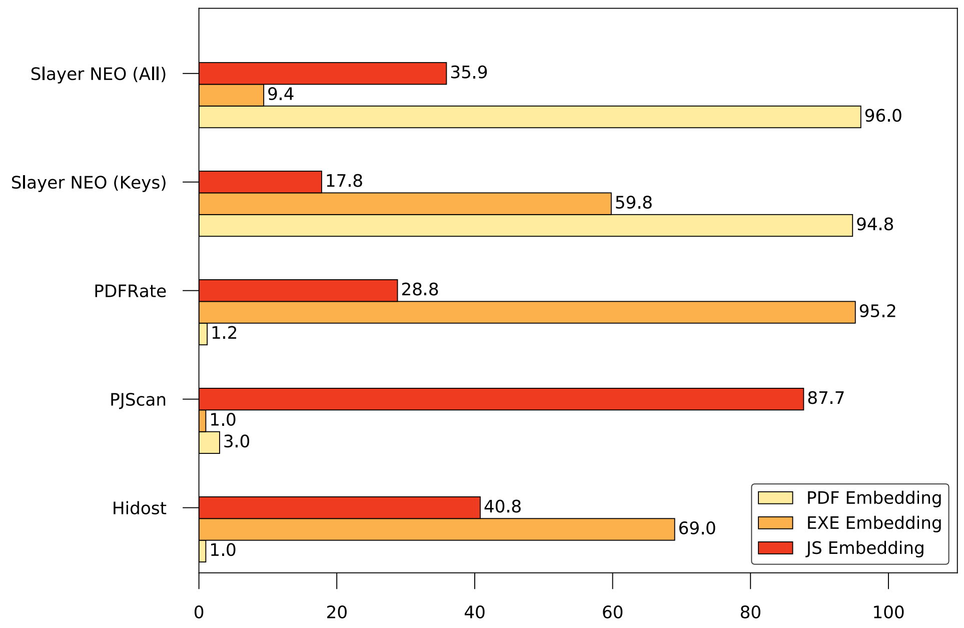

Fig. 4. Results attained by detection systems against reverse-mimicry attacks. Each system has been tested against 500 samples for each attack variant (1,500 samples in total). Results are reported by showing the percentage of detected samples.

malicious and benign files in the wild. Figure 4 reports the attained results. For each attack and system, we report the percentage of evasive samples that were detected. Note that `Slayer NEO` has been tested in two variants: the *keys* variant reflects the work made in Reference [64] (in which the system only operated by extracting keywords as features), while the *all* variant reflects the full functionality of the system, as tested in Reference [60].

Results clearly show that each system has its strengths and weaknesses. For example, `Slayer NEO` well performs against PDF embedding attacks due to its capability of extracting and analyzing embedded files, while `PJscan` is excellent at detecting `JS` embedding attacks, and `PDFRate-v1` provides reliable detection of `EXE` embedding. However, none of the tested systems can effectively detect all reverse-mimicry attacks.

Other black-Box Attacks. Other black-box attacks involve empirical attempts to exploit vulnerabilities of the components that belong to the target detectors, such as their parsers. This concept has been explored in a broader way by Carmony et al. [19], who showed that each parser (including advanced ones) on which PDF detectors are based could be evaded due to implementation bugs, design errors, omissions, and ambiguities. For this reason, attackers can be motivated to mostly target the pre-processing modules of detection systems, thus efficiently exploiting their vulnerabilities. Indeed, the authors created working proofs of concept with which they were able to evade both third-party and custom (`PDFRate`) parsers.

4.2.3 Practical Implementation. As already mentioned in Section 4.2 and in Table 8, a critical problem of adversarial attacks is the creation of the real attack samples starting from the evasive feature vectors (i.e., the so-called *inverse feature-mapping* problem). More specifically, the goal of the attacker is injecting/removing information into/from the PDF file while keeping its semantics intact (i.e., the file should work exactly like the unaltered version). However, due to the

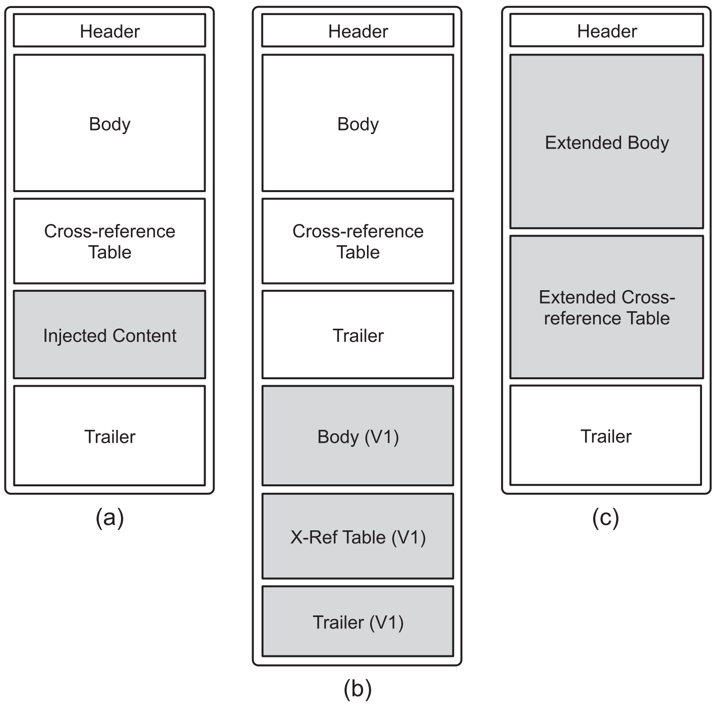

Fig. 5. Content injection in PDF files, performed according to three possible strategies: (a) Injection after X-Ref; (b) Versioning; and (c) Body Injection.

object-based structure of the PDF format (see Section 2), manipulating each feature value independently may not always be feasible. The corresponding file manipulations may compromise its malicious functionality, especially if the attack also requires removing content.

Table 8 showed which works implemented the real attack samples either by addressing the inverse feature-mapping problem or by directly manipulating the PDF malware sample. In both cases, three major strategies were used to inject material while minimizing the risk of compromising the intrusive functionality of PDF malware, as depicted in Figure 5 [62]. We now provide a comprehensive description of each technique by also referring to the works where they have been employed. Worth noting, these strategies can also be exploited to address the inverse feature-mapping problem and implement concrete attacks at the input level from the feature-level attack strategies proposed in References [8, 9, 22, 82, 95].

(a) Injection after X-Ref. In this strategy, objects are injected after the X-Ref table, and in particular after the EOF marker of the file. These objects are never parsed by Adobe Reader or similar, as their presence is not marked in the cross-reference table. This strategy is easy to implement, but it strongly relies on exploiting vulnerabilities of the parsing process. As clearly explained by Carmony et al. [19], all PDF detector parsers suffer from vulnerabilities, as none of them fully implements the PDF specifications released by Adobe. Such weaknesses are particularly evident in custom parsers, which may hide some strong misalignments to the behavior of Adobe Reader. However, such injection strategy can be made ineffective by merely patching the pre-processing module of the PDF malware detector to be consistent with Adobe Reader. Injection after X-Ref has been mostly employed by Šrndić and Laskov [96] to evade `PDFRate-v1`, which employs a customized parser. Albeit easy to counteract, this strategy has been used to simplify the injection of different types of structural features (for example, upper-/lower-case characters). The same strategy has been employed by Smutz et al. [83] for their experiments.

(b) Versioning. In this strategy, attackers use the *versioning* mechanism of the PDF file format, i.e., injecting a new body, X-Ref table, and trailer, as the user directly modified the file (e.g., by using an external tool (see Section 2). This strategy is more advanced than the previous one, as the cross-reference table parses the injected objects, and therefore they are considered as legitimate by the Reader itself. In this way, it is also straightforward to add embedded files or other malicious (or benign) contents that can be used as an aid to perpetuate a more effective attack. Such injection strategy can, however, be countered by extracting the content of each version of the PDF file and process it separately. Maiorca et al. [60–63] employed this strategy to generate EXE Embedding attacks (belonging to reverse mimicry). More specifically, they leveraged the `Metasploit` framework [75] to automatically generate the infected evasive samples. In this way, they showed that the versioning mechanism could be easily automatized by off-the-shelf tools to generate evasive variants.

(c) Body Injection. In this strategy, attackers directly operate on the existing PDF graph, adding new objects to the file body and re-arranging the X-Ref table accordingly. This strategy is more complicated to implement and to detect, as it seamlessly adds objects in a PDF file by reconstructing the objects and their position on the X-Ref Table. In this way, it is possible to manage and re-arrange the X-Ref table objects without corrupting the file [19]. Existing objects can also be modified by adding other name objects and rearranging the X-Ref table positions accordingly. Notably, it is essential to ensure that the embedded content (i.e., the exploitation code) is correctly executed when the merged PDF is opened. The correct execution of the embedded content is often not easy to achieve, as it requires injecting additional objects specifically for this purpose. This strategy was employed by Maiorca et al. [60–63] (and also used by Smutz et al. [83] and Carmony et al. [19]) to generate JS and PDF embedding attacks. In this way, the attacker can exactly choose in which part of the file body the malicious payload can be injected, thus making the automatic detection of such attacks harder.

An extension of the aforementioned body-injection strategy has been adopted by Xu et al. [102] to account also for object removal and replacement. In particular, after each manipulation of the file body (including object addition, removal, or replacement), the evasive sample is automatically tested on a Cuckoo sandbox. If the attempted change disrupts the functionality of the file (i.e., the PDF malware does not contact a malicious target URL anymore), the original file is restored. Albeit computationally expensive, this strategy allows one to precisely verify whether it is possible to perform such specific changes to the source PDF file.

4.3 Poisoning and Privacy Attacks Against PDF Malware Detectors

In this section, we discuss two other popular categories of attack defined in the literature of adversarial machine learning: poisoning and privacy attacks [16, 44]. Even though, to our knowledge, such attacks have never been considered in the context of PDF malware detection, we discuss here how they can be used to pose new threats to PDF malware detectors.

4.3.1 *Poisoning Attacks.*

Poisoning attacks aim to reduce the detection capabilities of PDF detectors by injecting wrongly labeled samples in the classifier training set. According to the taxonomy proposed in Section 4.1.4, we distinguish between three possible attacks:

Poisoning Availability Attacks. This attack can be carried out against online services that ask for the user feedback about the classification results (such as `PDFRate-v1`, which used to be available online). More specifically, the attacker can submit several malicious PDF files for the analysis.

When the system asks for feedback, she can intentionally claim that such samples are benign, hoping that the system gets retrained by including the wrongly labeled samples. If the attacker has no explicit control on the labeling process, she may construct benign samples that contain spurious malicious features, and malicious samples with benign content, in a way that preserves their original labeling. Similar attacks have been staged against anti-spam filters [44, 70] to eventually increase the classification error at test time and cause a denial of service to legitimate users.

Poisoning Integrity Attacks. This attack is similar to the one employed in poisoning availability. However, instead of aiming to increase the classification error at test time to cause a denial of service, the goal of the attacker here is to cause the misclassification of a specific subset of malicious PDF files at test time. For example, given a PDF malware sample that is correctly recognized by the target system, the attacker may start injecting benign samples with spurious malicious characteristics extracted from the malicious PDF file into the training set of the target classifier. Once updated, the target classifier may no longer correctly detect the given PDF malware sample. Overall, poisoning integrity attacks aim to facilitate evasion at test time.

Backdoor Attacks. The goal of this attack is the same as poisoning integrity, i.e., to facilitate evasion at test time, even though the attack strategy may be implemented in a different manner. In particular, the attacker may publicly release the backdoored model, which may be subsequently used in some publicly available online services or commercial tools. If this happens, the attacker can craft her malicious samples (including the backdoor activation signature) to bypass detection.

4.3.2 Privacy Attacks. Privacy attacks in the context of PDF malware detection may aim to primarily steal information about the classification model or the training data. They can be organized into three categories:

Model Stealing/Extraction Attacks. This attack can be performed on systems whose classification scores (or other information) are available [92]. The attacker can submit PDF files with progressive modifications to their structure (or scripting code) to infer which changes occur to the information retrieved from the models (in a similar fashion to what was performed by researchers in Reference [102]). In this way, the attacker may eventually reconstruct the detection model with high accuracy and sell an alternative online service at a cheaper price.

Model Inversion Attacks. The attack strategy is similar to model-stealing attacks, but the goal is reconstructing the employed training data [35]. This may violate user privacy if the attacker is able to reconstruct some benign PDF sample containing private information.

Membership Inference Attacks. This attack strategy is similar to the previous two cases, but the goal is to understand if a specific sample was used as part of the training data. For example, this can be useful to infer if the system was trained with PDF files that can be easily found on search engines, or if the system resorts to data available from publicly distributed malware datasets [69, 81].

5 OPEN ISSUES AND RESEARCH CHALLENGES

We discuss here the current open issues related to PDF malware detection and sketch some promising future research directions. More specifically, further research can be developed from two perspectives, both suggested from the research area of adversarial machine learning: (*i*) demonstrating novel *adversarial attacks* and potential threats against PDF malware detectors, and (*ii*) improving their *robustness* under attack by leveraging previous work on defensive mechanisms from adversarial machine learning. In the following, we discuss both perspectives.

5.1 Adversarial Attacks

Considering what we pointed out in Sections 4.2 and 4.3, it is evident that research on attacks against learning-based PDF malware detectors mainly focused on evasion. In particular, state-of-the-art work has been carried out by following two main directions: on the one hand, the creation of concrete, working adversarial examples by using empirical approaches, with the drawback of increasing the probability of failing the attacks due to too limited knowledge of the target system; on the other hand, the development of evasion algorithms, leading to approaches that allowed very efficient evasion without the creation of real samples (due to, for example, changes that could not be correctly performed in practice, such as deleting specific keywords). Notably, one critical problem to be solved is related to inverse feature mapping, i.e., creating real samples from the corresponding evasive feature vectors. Normally, to preserve the original behavior of the file, injecting information is typically safer than removing it. Nevertheless, such an action could compromise the overall file semantics and visualization properties. This effect could be triggered, e.g., by embedding font- or pages-related keywords in specific objects.

Recent work has demonstrated that it is possible to remove specific structural features (e.g., keywords) from PDF files without compromising their functionality [102]. However, there is still a lot of space for research on this topic. More specifically, attackers could focus on identifying a set of features that can be safely deleted (depending on the file context) or even replaced with equivalent ones. For example, one could remove every reference to `JavaScript` code and replace them with another language such as `ActionScript`. Concerning this aspect, it would also be intriguing to inspect the dependence between certain features (for example, deleting one keyword may force the attacker to also delete a second one). In this way, one may think of increasing the efficacy of gradient-descent algorithms by including a selection of erasable/replaceable information.

Finally, alternatives to evasion attacks can be explored. As stated in Section 4.3, poisoning and privacy attacks have yet to be carried out against PDF learners. One particularly interesting aspect is how such attacks can be useful in practice against current detection systems. For example, poisoning PDF detectors may compromise the performances of publicly available (or even open source) learning models. Likewise, model-stealing strategies can be employed to extract the characteristics of unknown detectors, leading to the development of more effective evasion attacks.

5.2 Robust Malware Detection

From what we pointed out in Sections 3 and 4, it is clear that every released detector features specific weaknesses that are either related to its parser, feature extractor, or classifier. However, while most state-of-the-art works focused on proposing evasion strategies, only a few pointed out and tested possible countermeasures against such attacks. In the following, we summarize the proposed mitigation approaches with the same organization proposed in Section 4.2.

Detection of Optimization-based Attacks. The only proposed approach to detect white- and gray-box gradient-based attacks against PDF files has been proposed by Smutz et al. [83], who proposed an improvement of the common tree-based ensemble detection mechanism used for PDF detectors. By considering the voting results of each tree-based component of the ensemble, the authors defined a region of *uncertainty* in which they classified evasion samples that obtained a score between specific thresholds. In this way—albeit the same samples were not explicitly regarded as malicious—the uncertain label would be enough to warn the user about possible evasion attempts. The attained results showed that `PDFRate-v2` performed significantly better than the previous `PDFRate-v1` at detecting the attack proposed by Šrndić and Laskov [96].

Detection of Heuristic-based Attacks. Concerning white-box mimicry, Smutz et al. [82] proposed a mitigation strategy in which the most discriminant features can be directly removed from the feature vector. However, such a choice can significantly impact the detection performances. To detect black-box reverse-mimicry attacks, Maiorca et al. [61–63] proposed a combination of features that are extracted both from the structure and the content (in terms of embedded code) of the file, along with the extraction and separate analysis of embedded PDF files. In this way, it is possible to significantly mitigate such attacks (especially PDF Embedding ones, which are entirely detected), although more than 30% of JS Embedding and EXE Embedding variants still managed to bypass detection [60–62]. For the same problem, Smutz et al. [83] employed the same mitigation strategy proposed for detecting optimization-based attacks and showed that the detection of reverse-mimicry attacks (in particular, JS Embedding) significantly improved. An exception to such an improved detection is the PDF embedding variant, which still manages to evade the system, as even PDFRate-v2 does not perform any analysis of possible embedded PDF payloads.

Despite previous efforts at detecting evasion attacks, it is clear that a bulletproof solution to detect such attacks has not yet been developed. Accordingly, we propose three research guidelines that can be further expanded and applied both against optimization- and heuristic-based attacks.

Parsing Reliability. Proper parsing should always include, among the others: *(i)* the extraction of the embedded content (that should be analyzed separately, either with the same or with different detectors); *(ii)* using consolidated parsers (e.g., third-party libraries or, in general, parsers whose specifications are the closest to the official specs); and *(iii)* robustness against malformed files (including benign ones) that may make the parser crash. Further research could focus on implementing these three characteristics and discussing their impact on the overall detection performances.

Feature Engineering. Following the results obtained by Maiorca et al. [61–63], it is clear that a proper feature engineering would drastically increase the robustness of the learning systems against black- and white-box attacks. For this reason, the key research point to be followed should be designing features that are hard to be modified from the perspective of the attacker. Dynamic features are generally more robust, as the attacker should change the file behavior to change the feature value. However, when it is not possible or feasible to analyze PDF files dynamically, combining different types of static features can be a winning solution that may significantly increase the effort that is required by the attacker to evade the target systems.

Robust and Explainable Learning Algorithms. Research should push on developing algorithms that are more robust against perturbations made by attackers. To this end, explicit knowledge of the attack model has to be considered during the design of the learning algorithm. One possibility is to use robust optimization algorithms or, more generally, game-theoretical models. Such models have been widely studied in the literature of adversarial machine learning [17, 38], largely before the introduction of *adversarial training* [39], which essentially follows the same underlying idea. Under this game-theoretical learning framework, the attacker and the classifier maximize different objective functions.[10] In particular, while the attacker manipulates data within certain bounds to evade detection, the classifier is iteratively retrained to correctly detect the manipulated samples [16, 17, 38, 77]. We argue that game-theoretical models may be helpful to improve robustness of PDF malware detectors. Explainable machine-learning algorithms may provide another useful asset for system designers to analyze the trained models and understand their weaknesses. This approach has been recently adopted to inspect the security properties of learning systems and highlight their vulnerabilities [5, 27, 43, 67, 76]. In particular, it has been al-

[10]For the sake of completeness, it is worth pointing out that, in robust optimization, they maximize the same objective with opposite sign, yielding a zero-sum game.

ready observed (e.g., in spam filtering [11, 50] and Android malware detection [28]) that learning algorithms may tend to overemphasize some features, and that this facilitates evasion by a skilled attacker who carefully manipulates only those feature values. Developing specific learning approaches that distribute feature importance more evenly can significantly improve robustness to such adversarial evasion attempts [11, 28, 50, 67]. This has been clearly demonstrated by Demontis et al. [28] in the context of adversarial Android malware detection. In that work, the authors have shown that classifiers trained using a principled approach based on robust optimization eventually provide more evenly distributed feature weights and much stronger robustness guarantees (even than the robust deep networks developed for the same task in Reference [41]). Another defensive mechanism against evasion attacks consists of *rejecting* anomalous samples at test time, similarly to the uncertainty-region approach defined by Smutz et al. [83]. More specifically, when a test sample is assigned an anomalous score in comparison to the scores typically assigned to benign or malware samples, it can be flagged as anomalous or classified as suspicious, requiring further manual inspection (see, e.g., the work in Reference [66] for an example on adversarial robot vision). Defensive mechanisms have also been proposed against poisoning and privacy attacks based on training data sanitization and robust learning, respectively, and on leveraging differentially private mechanisms for learning private classifiers (we refer the reader to Reference [16] for further details).

We finally argue that the aforementioned guidelines are not only useful for PDF malware detectors, but they can be extended to other malware detection systems that may be targeted by adversarial attacks. We believe that a correct application of the principles described above can be genuinely beneficial and can constitute a substantial aid to increase security of such systems.

6 CONCLUSIONS

In this article, we presented a survey of malicious PDF detection in adversarial environments featuring multi-folded contributions. First, we described how malicious PDF files perform their attacks in the wild. Accordingly, we described all machine learning–based solutions for PDF malware detection that have been proposed in the past decade. Notably, this part differentiates from previously proposed surveys, such as the ones by Nissim et al. [71] and Elingiusti et al. [31], which did not focus on those system components that are crucial to understanding adversarial attacks. For example, our work provided a deep insight into the pre-processing of PDF files, which has been exploited by many adversarial attacks. Second, we provided a comprehensive overview of the adversarial attacks that have been carried out against PDF malware detectors. In particular, we categorized the actual adversarial attacks against PDF detectors under a unifying framework and described them by also sketching possible novel strategies that can be employed by attackers. To this end, we employed a methodology similar to previous work on command-and-control botnets [37]. However, PDF malware detection is clearly a different task and, to the best of our knowledge, this is the first work that thoroughly described adversarial machine learning applied to this field. The significant adversarial-related traits of this survey also completely distinguish it from other works on data mining for malware detection; for example, the work in Reference [103] is not focused on the adversarial aspects of the problem. It is worth noting that Cuan et al. [24] pointed out some basics aspects of how adversarial machine learning has been applied to the detection of malicious PDF files. However, our work expands and discusses such a problem in much wider detail. Finally, we discussed existing mitigation strategies and sketched new research directions that could allow detecting not only current adversarial attacks but also novel ones.

Notably, the main goal of this work is that of discussing how PDF malware analysis has brought, over recent years, significant advancements in understanding how adversarial machine learning could be applied to malware detection. The recent results described in this work have been beneficial in both research fields. On the one hand, research demonstrated that adversarial attacks constitute a concrete, real emerging security threat that can be extremely dangerous, as machine learning is now widely employed even by standard anti-malware solutions. On the other hand, discoveries concerning adversarial attacks pushed developers and security analysts to develop better protections and to explore novel information about malware that can be useful for classification. We believe that future work should focus on a more rigorous application of security-by-design principles when building detection systems. In this way, it will be possible to offer real, active mitigation of the numerous evasion attacks that are being progressively used in the wild.

REFERENCES

[1] Adobe. 2006. *PDF Reference. Adobe Portable Document Format Version 1.7.* https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdf_reference_archive/pdf_reference_1-7.pdf.

[2] Adobe. 2007. *JavaScript for Acrobat API Reference.* https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/js_api_reference.pdf.

[3] Adobe. 2008. *Adobe Supplement to ISO 32000.* https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/adobe_supplement_iso32000.pdf.

[4] Anish Athalye, Nicholas Carlini, and David A. Wagner. 2018. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *Proceedings of the International Conference on Machine Learning (ICML'18) (JMLR Workshop and Conference Proceedings)*, Vol. 80. JMLR.org, 274–283.

[5] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. 2010. How to explain individual classification decisions. *J. Machine Learn. Res.* 11 (2010), 1803–1831.

[6] Marco Barreno, Blaine Nelson, Anthony Joseph, and J. Tygar. 2010. The security of machine learning. *Machine Learn.* 81 (2010), 121–148. Issue 2.

[7] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. 2006. Can machine learning be secure? In *Proceedings of the ACM Symp. Information, Computer and Comm. Sec. (ASIACCS'06).* ACM, New York, NY, 16–25.

[8] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Part III (LNCS)*, Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný (Eds.), Vol. 8190. Springer Berlin Heidelberg, 387–402.

[9] Battista Biggio, Igino Corona, Blaine Nelson, Benjamin I. P. Rubinstein, Davide Maiorca, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. 2014. Security Evaluation of Support Vector Machines in Adversarial Environments. In *Support Vector Machines Applications*, Yunqian Ma and Guodong Guo (Eds.). Springer International Publishing, Cham, 105–153. DOI: https://doi.org/10.1007/978-3-319-02300-7_4

[10] Battista Biggio, Luca Didaci, Giorgio Fumera, and Fabio Roli. 2013. Poisoning attacks to compromise face templates. In *Proceedings of the 6th IAPR International Conference on Biometrics (ICB'13).* 1–7.

[11] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2010. Multiple classifier systems for robust classifier design in adversarial environments. *Int. J. Machine Learn. Cyber.* 1, 1 (2010), 27–41.

[12] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2014. Pattern recognition systems under attack: Design issues and research challenges. *Int. J. Pattern Recog. Artific. Intell.* 28, 7 (2014), 1460002.

[13] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2014. Security evaluation of pattern classifiers under attack. *IEEE Trans. Knowl. Data Eng.* 26, 4 (Apr. 2014), 984–996.

[14] Battista Biggio, Giorgio Fumera, Fabio Roli, and Luca Didaci. 2012. Poisoning adaptive biometric systems. In *Structural, Syntactic, and Statistical Pattern Recognition*, Georgy Gimel'farb, Edwin Hancock, Atsushi Imiya, Arjan Kuijper, Mineichi Kudo, Shinichiro Omachi, Terry Windeatt, and Keiji Yamada (Eds.). Lecture Notes in Computer Science, Vol. 7626. Springer Berlin, 417–425. DOI: http://dx.doi.org/10.1007/978-3-642-34166-3_46

[15] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines, In *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*, John Langford and Joelle Pineau (Eds.). 1807–1814.

[16] Battista Biggio and Fabio Roli. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recog.* 84 (2018), 317–331. DOI: https://doi.org/10.1016/j.patcog.2018.07.023

[17] Michael Brückner, Christian Kanzow, and Tobias Scheffer. 2012. Static prediction games for adversarial learning problems. *J. Machine Learn. Res.* 13, 1 (Sept. 2012), 2617–2654. Retrieved from: http://dl.acm.org/citation.cfm?id=2503308.2503326.

[18] Nicholas Carlini and David A. Wagner. 2017. Towards evaluating the robustness of neural networks. In *Proceedings of the IEEE Symposium on Security and Privacy.* IEEE Computer Society, 39–57.

[19] Curtis Carmony, Mu Zhang, Xunchao Hu, Abhishek Vasisht Bhaskar, and Heng Yin. 2016. Extract me if you can: Abusing PDF parsers in malware detectors. In *Proceedings of the 23rd Network & Distributed System Security Symposium (NDSS'16)*.

[20] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. 2017. ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec'17)*. ACM, 15–26.

[21] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. Retrieved from: *ArXiv E-prints* abs/1712.05526 (2017).

[22] Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. 2014. Lux0R: Detection of malicious PDF-embedded JavaScript code through discriminant analysis of API references. In *Proceedings of the Workshop on Artificial Intelligent and Security Workshop (AISec'14)*. ACM, 47–57.

[23] Marco Cova, Christopher Kruegel, and Giovanni Vigna. 2010. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*. ACM, 281–290. DOI: https://doi.org/10.1145/1772690.1772720

[24] Bonan Cuan, Aliénor Damien, Claire Delaplace, and Mathieu Valois. 2018. Malware detection in PDF files using machine learning. In *Proceeding of the 15th International Conference on Security and Cryptography (SECRYPT'18)*.

[25] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. 2004. Adversarial classification. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*. 99–108.

[26] Hung Dang, Yue Huang, and Ee-Chien Chang. 2017. Evading classifiers by morphing in the dark. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*. ACM, 119–133.

[27] Luca Demetrio, Battista Biggio, Giovanni Lagorio, Fabio Roli, and Alessandro Armando. 2019. Explaining vulnerabilities of deep learning to adversarial malware binaries. In *Proceedings of the 3rd Italian Conference on Cyber Security (ITASEC'19)*, Vol. 2315. CEUR Workshop Proceedings.

[28] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2019. Yes, machine learning can be more secure! A case study on Android malware detection. *IEEE Trans. Depend. Sec. Comput.* 16, 4 (2019), 711–724.

[29] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. 2018. Why do adversarial attacks transfer? Explaining transferability of evasion and poisoning attacks. Retrieved from: *Arxiv E-prints*, Article arXiv:1809.02861.

[30] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Xiaolin Hu, and Jun Zhu. 2018. Boosting adversarial examples with momentum. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'18)*.

[31] Michele Elingiusti, Leonardo Aniello, Leonardo Querzoni, and Roberto Baldoni. 2018. *PDF-Malware Detection: A Survey and Taxonomy of Current Techniques*. Springer International Publishing, Cham, 169–191. DOI: https://doi.org/10.1007/978-3-319-73951-9_9

[32] ESET. 2018. A tale of two zero-days. Retrieved from: https://www.welivesecurity.com/2018/05/15/tale-two-zero-days/.

[33] Jose Miguel Esparza. 2017. PeePDF. Retrieved from: http://eternal-todo.com/tools/peepdf-pdf-analysis-tool.

[34] Fortinet. 2016. Analysis of CVE-2016-4203—Adobe Acrobat and Reader CoolType Handling Heap Overflow Vulnerability. Retrieved from: https://www.fortinet.com/blog/threat-research/analysis-of-cve-2016-4203-adobe-acrobat-and-reader-cooltype-handling-heap-overflow-vulnerability.html.

[35] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. ACM, 1322–1333.

[36] FreeDesktop.org. 2018. Poppler. Retrieved from: https://poppler.freedesktop.org/.

[37] Joseph Gardiner and Shishir Nagaraja. 2016. On the security of machine learning in malware C&C detection: A survey. *ACM Comput. Surv.* 49, 3, Article 59 (Dec. 2016), 39 pages. DOI: https://doi.org/10.1145/3003816

[38] Amir Globerson and Sam T. Roweis. 2006. Nightmare at test time: Robust learning by feature deletion. In *Proceedings of the 23rd International Conference on Machine Learning*, William W. Cohen and Andrew Moore (Eds.), Vol. 148. ACM, 353–360.

[39] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations*.

[40] Google. 2018. Virustotal. Retrieved from: http://www.virustotal.com.

[41] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. 2017. Adversarial examples for malware detection. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS'17) (LNCS)*, Vol. 10493. Springer, 62–79.

[42] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying vulnerabilities in the machine learning model supply chain. In *Proceedings of the NIPS Workshop on Machine Learning and Computer Security*, Vol. abs/1708.06733.

[43] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2019. A survey of methods for explaining black box models. *ACM Comput. Surv.* 51, 5 (2019), 93:1–93:42.

[44] L. Huang, A. D. Joseph, B. Nelson, B. Rubinstein, and J. D. Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Artificial Intelligence and Security (AISec'11)*. 43–57.

[45] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. 2018. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'18)*. IEEE CS, 931–947. DOI: https://doi.org/10.1109/SP.2018.00057

[46] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. 2018. Model-reuse attacks on deep learning systems. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*. ACM, 349–363.

[47] Kaspersky. 2017. *Machine Learning for Malware Detection.* https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf.

[48] Marius Kloft and Pavel Laskov. 2012. Security analysis of online centroid anomaly detection. *J. Machine Learn. Res.* 13 (2012), 3647–3690.

[49] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the International Conference on Machine Learning (ICML'17)*.

[50] Aleksander Kolcz and Choon Hui Teo. 2009. Feature weighting for improved classifier robustness. In *Proceedings of the 6th Conference on Email and Anti-Spam (CEAS'09)*.

[51] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert, and Fabio Roli. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *Proceedings of the 26th European Signal Processing Conference (EUSIPCO'18)*. IEEE, 533–537.

[52] Sogeti ESEC Lab. 2015. Origami Framework. Retrieved from: http://esec-lab.sogeti.com/pages/origami.html.

[53] Pavel Laskov and Nedim Šrndić. 2011. Static detection of malicious JavaScript-bearing PDF documents. In *Proceedings of the 27th Computer Security Applications Conference (ACSAC'11)*. ACM, 373–382. DOI: https://doi.org/10.1145/2076732.2076785

[54] Daiping Liu, Haining Wang, and Angelos Stavrou. 2014. Detecting malicious JavaScript in PDF through document instrumentation. In *Proceedings of the 44th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'14)*. IEEE Computer Society, 100–111. DOI: https://doi.org/10.1109/DSN.2014.92

[55] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2017. Delving into transferable adversarial examples and black-box attacks. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

[56] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning attack on neural networks. In *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS'18)*.

[57] Daniel Lowd and Christopher Meek. 2005. Adversarial learning. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)*. ACM Press, 641–647.

[58] Xun Lu, Jianwei Zhuge, Ruoyu Wang, Yinzhi Cao, and Yan Chen. 2013. De-obfuscation and detection of malicious PDF files with high accuracy. In *Proceedings of the 46th Hawaii International Conference on System Sciences*. 4890–4899. DOI: https://doi.org/10.1109/HICSS.2013.166

[59] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*.

[60] Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giacinto. 2015. An evasion resilient approach to the detection of malicious PDF files. In *Information Systems Security and Privacy*, Olivier Camp, Edgar Weippl, Christophe Bidan, and Esma Aïmeur (Eds.). Springer International Publishing, Cham, 68–85.

[61] Davide Maiorca, Davide Ariu, Igino Corona, and Giorgio Giacinto. 2015. A structural and content-based approach for a precise and robust detection of malicious PDF files. In *Proceedings of the 1st International Conference on Information Systems Security and Privacy (ICISSP'15)*. 27–36. DOI: https://doi.org/10.5220/0005264400270036

[62] Davide Maiorca and Battista Biggio. 2019. Digital investigation of PDF files: Unveiling traces of embedded malware. *IEEE Secur. Privacy* 17, 01 (Jan. 2019), 63–71.

[63] Davide Maiorca, Igino Corona, and Giorgio Giacinto. 2013. Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious PDF files detection. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS'13)*. ACM, 119–130.

[64] Davide Maiorca, Giorgio Giacinto, and Igino Corona. 2012. A pattern recognition system for malicious PDF files detection. In *Machine Learning and Data Mining in Pattern Recognition (Lecture Notes in Computer Science)*, Petra Perner (Ed.), Vol. 7376. Springer Berlin, 510–524.

[65] Shike Mei and Xiaojin Zhu. 2015. Using machine teaching to identify optimal training-set attacks on machine learners. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15).*

[66] Marco Melis, Ambra Demontis, Battista Biggio, Gavin Brown, Giorgio Fumera, and Fabio Roli. 2017. Is deep learning safe for robot vision? Adversarial examples against the iCub humanoid. In *Proceedings of the ICCVW Vision in Practice on Autonomous Robots (ViPAR'17).* IEEE, 751–759.

[67] Marco Melis, Davide Maiorca, Battista Biggio, Giorgio Giacinto, and Fabio Roli. 2018. Explaining black-box Android malware detection. In *Proceedings of the 26th European Signal Processing Conference (EUSIPCO'18).* IEEE, 524–528.

[68] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. 2017. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (AISec'17)*, Bhavani M. Thuraisingham, Battista Biggio, David Mandell Freeman, Brad Miller, and Arunesh Sinha (Eds.). ACM, 27–38.

[69] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Machine learning with membership privacy using adversarial regularization. In *Proceedings of the ACM Conference on Computer and Communications Security.* ACM, 634–646.

[70] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. 2008. Exploiting machine learning to subvert your spam filter. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET'08).* USENIX Association, 1–9.

[71] Nir Nissim, Aviad Cohen, Chanan Glezer, and Yuval Elovici. 2015. Detection of malicious PDF files and directions for enhancements: A state-of-the art survey. *Comput. Secur.* 48 (2015), 246–266.

[72] Sentinel One. 2018. SentinelOne Detects New Malicious PDF File. Retrieved from: https://www.sentinelone.com/blog/sentinelone-detects-new-malicious-pdf-file/.

[73] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the ACM on Asia Conference on Computer and Communications Security (ASIA CCS'17).* ACM, 506–519.

[74] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy.* IEEE, 372–387.

[75] Rapid7. 2019. Metasploit framework. Retrieved from: https://www.metasploit.com/.

[76] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16).* ACM, 1135–1144.

[77] S. Rota Bulò, B. Biggio, I. Pillai, M. Pelillo, and F. Roli. 2017. Randomized prediction games for adversarial machine learning. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 11 (2017), 2466–2478.

[78] Florian Schmitt, Jan Gassen, and Elmar Gerhards-Padilla. 2012. PDF scrutinizer: Detecting JavaScript-based attacks in PDF documents. In *Proceedings of the 10th International Conference on Privacy, Security and Trust (PST)*, Vol. 00. 104–111. DOI: https://doi.org/10.1109/PST.2012.6297926

[79] Offensive Security. 2018. Exploit Database. Retrieved from: https://www.exploit-db.com/.

[80] M. Zubair Shafiq, Syed Ali Khayam, and Muddassar Farooq. 2008. Embedded malware detection using Markov n-grams. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'08).* Springer-Verlag, Berlin, 88–107. DOI: https://doi.org/10.1007/978-3-540-70542-0_5

[81] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'17).* 3–18.

[82] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF detection using metadata and structural features. In *Proceedings of the 28th Computer Security Applications Conference (ACSAC'12).* ACM, 239–248. DOI: https://doi.org/10.1145/2420950.2420987

[83] Charles Smutz and Angelos Stavrou. 2016. When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS'16).* Retrieved from: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/when-tree-falls-using-diversity-ensemble-classifiers-identify-evasion-malware-detectors.pdf.

[84] Kevin Z. Snow, Srinivas Krishnan, Fabian Monrose, and Niels Provos. 2011. SHELLOS: Enabling fast detection and forensic analysis of code injection attacks. In *Proceedings of the 20th USENIX Conference on Security (SEC'11).* USENIX Association, 9–9. Retrieved from: http://dl.acm.org/citation.cfm?id=2028067.2028076.

[85] Nedim Šrndić and Pavel Laskov. 2016. Hidost: A static machine-learning-based detector of malicious files. *EURASIP J. Inform. Sec.* 2016, 1 (26 Sep 2016), 22. DOI: https://doi.org/10.1186/s13635-016-0045-0

[86] Didier Stevens. 2008. PDF Tools. Retrieved from: http://blog.didierstevens.com/programs/pdf-tools.

[87] Symantec. 2018. *Internet Security Threat Report (Vol. 23).* https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf.

[88] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations*. Retrieved from: http://arxiv.org/abs/1312.6199.

[89] S. Momina Tabish, M. Zubair Shafiq, and Muddassar Farooq. 2009. Malware detection using statistical analysis of byte-level file content. In *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*.

[90] Trevor Tonn and Kiran Bandla. 2013. PhoneyPDF. Retrieved from: https://github.com/kbandla/phoneypdf.

[91] Malware Tracker. 2018. PDF Current Threats. Retrieved from: https://www.malwaretracker.com/pdfthreat.php.

[92] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction APIs. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security'16)*. USENIX Association, 601–618.

[93] Zacharias Tzermias, Giorgos Sykiotakis, Michalis Polychronakis, and Evangelos P. Markatos. 2011. Combining static and dynamic analysis for the detection of malicious documents. In *Proceedings of the 4th European Workshop on System Security (EUROSEC'11)*. ACM, Article 4, 6 pages. DOI: https://doi.org/10.1145/1972551.1972555

[94] Cristina Vatamanu, Dragoş Gavriluţ, and Răzvan Benchea. 2012. A practical approach on clustering malicious PDF documents. *J. Comput. Virol.* 8, 4 (Nov. 2012), 151–163. DOI: https://doi.org/10.1007/s11416-012-0166-z

[95] Nedim Šrndić and Pavel Laskov. 2013. Detection of malicious PDF files based on hierarchical document structure. In *Proceedings of the 20th Network & Distributed System Security Symposium (NDSS'13)*.

[96] Nedim Šrndic and Pavel Laskov. 2014. Practical evasion of a learning-based classifier: A case study. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'14)*. IEEE Computer Society, 197–211. DOI: https://doi.org/10.1109/SP.2014.20

[97] Nedim Šrndic and Pavel Laskov. 2014. Practical evasion of a learning-based classifier: A case study. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'14)*. IEEE CS, 197–211.

[98] VulDB. 2018. The Crowd-Based Vulnerability Database. Retrieved from: https://vuldb.com.

[99] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, Alexander G. Ororbia, Xinyu Xing, Xue Liu, and C. Lee Giles. 2017. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'17)*, Vol. Part F129685. Association for Computing Machinery, 1145–1153. DOI: https://doi.org/10.1145/3097983.3098158

[100] Carsten Willems, Felix C. Freiling, and Thorsten Holz. 2012. Using memory management to detect and extract illegitimate code for malware analysis. In *Proceedings of the 28th Computer Security Applications Conference (ACSAC'12)*. ACM, 179–188. DOI: https://doi.org/10.1145/2420950.2420979

[101] Meng Xu and Taesoo Kim. 2017. PlatPal: Detecting malicious documents with platform diversity. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security'17)*. USENIX Association, 271–287. Retrieved from: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/xu-meng.

[102] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically evading classifiers. In *Proceedings of the 23rd Network & Distributed System Security Symposium (NDSS'16)*.

[103] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A survey on malware detection using data mining techniques. *ACM Comput. Surv.* 50, 3, Article 41 (June 2017), 40 pages. DOI: https://doi.org/10.1145/3073559

Received October 2018; revised April 2019; accepted May 2019