# Digital Investigation of PDF Files

## Unveiling Traces of Embedded Malware

**Davide Maiorca and Battista Biggio** | University of Cagliari

**As it continues to proliferate, malware has shown increasing sophistication, and PDF malware is a major threat on the cybersecurity landscape. We provide an overview of current attack techniques used to convey PDF malware and the analysis tools that support digital forensic investigations.**

In recent years, the number of services available on the Internet, along with the number of interconnected users, has rapidly increased. This has revolutionized the way society is organized, facilitating how we communicate, work, and perform our daily activities. However, this rapid expansion of the Internet has also resulted in severe drawbacks. The first relates to the fact that we are essentially dipped into a liquid state where a vast amount of our personal data—a valuable asset both for companies and cybercriminals—is provided in a seamless manner to third-party services, with no guarantees regarding how it will be managed and stored. Second, the proliferation of web services has also drastically increased the number of vulnerable applications exploitable by cybercriminals. Cybercrime has become a very profitable activity, and cybercriminals reinvest profits made on the black markets or other illicit activities (e.g., violated online bank accounts) to improve their illegal business. The fact that attackers are economically motivated and constantly aim to mislead current cybersecurity systems is the main motivation for the constant evolution, sophistication, and variability of malware and other scams perpetuated over the Internet.

PDF documents have been among the major vectors used to convey malware, thanks to the flexibility of their structure and the fact that they support embedding different kinds of content, ranging from JavaScript to ActionScript codes. Although Microsoft Office macro-based attacks now play a major role in the diffusion of malware, critical vulnerabilities are still being publicly disclosed for Adobe Reader [e.g., see Common Vulnerabilities and Exposures (CVE)-2017-3010 and CVE-2016-1009 from the National Vulnerability Database]. PDF malware thus remains a potential, serious threat for Internet users, as also witnessed by recent research.[13,14]

Malware embedding in PDF files can be largely automatized with state-of-the-art tools like Metasploit. PDF files also support embedding of obfuscated or encrypted content, which can be leveraged by an attacker to increase the probability of evading anti-malware mechanisms. Another reason behind the proliferation of malicious PDF files is that, normally, unexperienced users receiving such files (e.g., as attachments to scam emails) tend to trust and execute them, as they are not commonly known as potential malware vectors.

*Digital Object Identifier 10.1109/MSEC.2018.2875879*
*Date of publication: 20 March 2019*

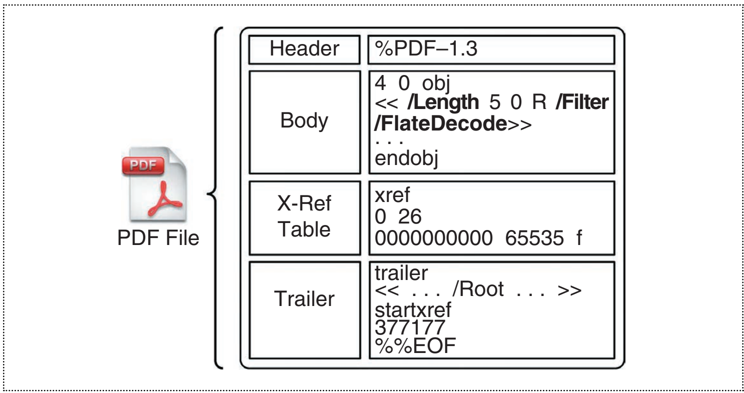


**Figure 1.** The PDF file structure, with examples of the header, body, x-ref table, and trailer contents. Object names (i.e., keywords) are highlighted in bold.

Due to the inherent flexibility and complexity of the format and the variability of the attacks, effectively analyzing and recognizing malicious PDF files has become a compelling challenge, especially from the viewpoint of a forensic analyst. For these reasons, machine learning has been exploited as a key component in the development of more recent PDF malware detection systems, either to prevent infection of a targeted machine or to help the analyst during a forensic investigation (after the incident).[1–8] Nevertheless, as machine learning has not been designed to operate against intelligent attackers, it exposes specific vulnerabilities that can be exploited to evade detection.

In this article, we first provide an overview of the PDF and the current attacks used to convey PDF malware through concrete attack examples collected in the wild. We describe how to perform a forensic analysis of a PDF file to find evidence of embedded malware using state-of-the-art software tools. We then discuss some recent PDF malware detection tools based on machine learning that can support digital forensic analyses, identifying suspicious files before a deeper, more detailed manual investigation is launched. We discuss PDF limitations and related open issues, especially in terms of the exploitation of their vulnerabilities to potentially mislead subsequent forensic analyses. Finally, we suggest guidelines for improving the performance of such systems under attack and then sketch promising research directions.

## The PDF

The PDF is one of the most used formats to render documents. Due to the support for third-party technologies such as JavaScript and ActionScript, the PDF is widely used not only for visualizing text but also for rendering images, compiling forms, and showing animations. The typical structure of a PDF file is depicted in Figure 1. It consists of four parts:

1) the header, containing information about the PDF file version
2) the body, containing a number of objects that define the operations performed by the file and the embedded data (e.g., text, images, and scripting code)
3) the cross-reference (x-ref) table, listing the offset of each object inside the file to be rendered by the reader
4) the trailer, i.e., a special object that describes, among others, the first object to be rendered by the reader, identified by the name object */Root*.

Technically, a PDF file can be seen as a graph of objects that instructs the reader about the operations it must do to visualize the file content for the user. The PDF supports eight types of objects:

- *boolean*, i.e., a variable that can be `True` or `False`
- *numeric*, i.e., a real or integer value
- *string*, i.e., a sequence of characters between parentheses, () or a sequence of hexadecimal characters between angle brackets, <>
- *name*, i.e., a literal sequence of characters that starts with a forward slash, /
- *array*, i.e., a sequence of objects between square brackets, []
- *dictionary*, i.e., an object comprising a sequence of key-value pairs, enclosed by double angle brackets, << >> (e.g., the *trailer* object is a dictionary)
- *stream*, i.e., a special object consisting of a dictionary and a sequence of data (typically, compressed text or images) introduced by the keyword stream.

These objects are divided into two categories. Objects marked by a number (introduced by the string *objNum 0 obj*) are called *indirect*, whereas objects not marked by a number are called *direct*. Indirect objects are typically dictionaries and can be referenced by other objects with the string *objNum 0 Ref*. An example of an indirect object introduced by the string *4 0 obj* is shown in Figure 1. In this case, the keyword `/Length` introduces the size of the object, whose value is contained in object 5 (this reference is defined by *5 0 R*). The remaining two keywords define the characteristic of the object, which, in this case, contains information about the filter used for data compression (`/FlateDecode`). The PDF graph mainly contains indirect objects.

When a reader renders a PDF file, it starts from the trailer object and parses each indirect object (referenced by the x-ref table), decompressing its data. In this way, all pages, text, images, and scripting code are progressively shown.

## PDF Malware

The capability of embedding different kinds of content does not only makes the PDF a convenient way of legitimately sharing information, it also gives attackers the possibility of exploiting a larger number of potential vulnerabilities. In fact, PDF malware is multifaceted, conceived to exploit the flexible nature of the PDF. Typically, JavaScript code, encoded streams, and embedded objects (e.g., images, ActionScript code) are used to exploit a vulnerability of the PDF reader and subsequently allow execution of remote code. In the following, we briefly discuss some popular examples of attacks in which an embedded object [respectively, an image, an executable, and a ShockWave Flash file (SWF)] is used to exploit a vulnerability of the PDF reader.

The first example exploits the so-called Adobe Reader BMP/RLE heap corruption vulnerability (CVE-2013-2729) to download and install malware from a remote website(http://eternal-todo.com/blog/cve-2013-2729-exploit-zeusp2p-gameover). In this case, the malicious PDF file contains a form with an encoded bitmap image. When the PDF file is opened, the image is automatically decoded, causing a heap overflow that allows execution of remote code.

Another example, reported by Contagio in 2010, shows how to execute binary code by simply opening a PDF file (CVE-2010-1240; http://contagiodump.blogspot.it/2010/08/cve-2010-1240-with-zeus-trojan.html). A code excerpt of the embedded object that implements this attack follows:

```
155 0 obj
<<
/Type/Action/S/Launch/Win
<<
/F (cmd.exe)
/P (/c echo Dim BinaryStream > vbs1.vbs
&& echo Set BinaryStream =
CreateObject("ADODB.Stream") >>
… >>
endobj
```

This object executes the (Windows) command prompt (`cmd.exe`) and uses it to run a Visual Basic script that retrieves and executes the Zeus trojan. Notably, in subsequent versions of Adobe Reader, execution of binary files has been inhibited to limit this kind of exploitation. Despite this, and even if this attack is almost nine years old, only 33 of the 55 antimalware systems used in `VirusTotal` correctly detect this file as malicious.

Malicious SWF files and ActionScript code can also be embedded in PDF files to exploit vulnerabilities of the Flash interpreter used by Adobe Reader. An example is the zero-day vulnerability discovered in 2010 (CVE-2010-1297; https://blog.zynamics.com/2010/06/09/analyzing-the-currently-exploited-0-day-for-adobe-reader-and-adobe-flash/), which was exploited through the execution of a malicious ActionScript code fragment contained in an embedded SWF file. After exploitation, the infection of the victim machine was completed by running an executable file, also stored as an encrypted stream in the PDF file, which eventually dropped other malware from malicious websites.

Besides embedding external objects, using malicious JavaScript code constitutes the prominent way to attack Adobe Reader. In particular, the goal of the exploit is typically to bypass memory protections such as data execution prevention and address space layout randomization by resorting to a combination of heap spraying and return-oriented programming (ROP) gadgets. The main idea here is that the attacker fills the heap with multiple replicas of no-operation sleds and shellcode (typically built through ROP gadgets, instructions belonging to existing, legitimate libraries that can be combined to build malicious routines). This is done to increase the probability that, after memory corruption, the execution of the process is redirected to the malicious code. An example of this exploit procedure is the CVE-2014-0496 vulnerability, whose full description was reported previously.[8]

The aforementioned examples of attack only constitute a small excerpt of the set of possibilities that an attacker has to exploit the vulnerabilities of PDF readers. Nonetheless, they clearly show how sophisticated and different such attacks can be, also highlighting the complexity of the detection task.

## Forensic Analysis of PDF Malware

From a forensic perspective and assuming the infection started from a PDF file, it is essential to depict a basic road map the analyst can follow to identify the suspicious PDF files. They can be detected (also after infection) by the machine-learning approaches described in the remainder of this article or identified through some other source of information (e.g., by discussing with the victim the possibility of being phished by a scam email). Then, their content can be analyzed to identify the actions performed by the malicious code. Accordingly, the analyst is required to first find the *suspicious indirect objects* (i.e., the malicious scripting code or files embedded in the PDF) that are responsible for the malware infection. To this end, analysts might employ three different approaches, detailed here.

### Keyword-Based Analysis

The goal here is to extract the content (keywords) of indirect objects to identify the actions performed by the file; e.g., if the keyword `/JavaScript` is present, the file contains some scripting code. Such analysis does not typically decompress the streams related to the object, but it can give the analyst a quick overview of which file parts to analyze in more detail. Normally, if no suspicious keyword is present in the file, then the file can be considered as safe.

`PDFiD` (https://blog.didierstevens.com/2009/03/31/pdfid/), which extracts name objects, is a forensic tool for PDF files that can greatly aid this approach. It performs textual analysis of all the dictionaries included in the file, such as objects that can be easily visualized with a simple text editor (e.g., `Notepad`). The result is a list of keyword objects, along with their occurrence in the file. However, such a tool can be easily deceived. First, there is no control on how the objects are connected to each other. This means that the tool can report objects that are never parsed by the reader. Moreover, the tool does not consider the global structure of the PDF file. Hence, it can extract objects from positions in which they could never be parsed by the reader (as happened previously[9]). For these reasons, the results provided by the `PDFiD` should be further confirmed by other tools/approaches.

### Tree-Based Analysis

Here, the goal is to reconstruct the PDF file *tree*, i.e., the interconnections among its objects. `PeePDF` (http://eternal-todo.com/tools/peepdf-pdf-analysis-tool) is a publicly available software that performs this operation automatically. In particular, its analysis is performed as follows: the system first looks for the trailer object (containing the `/Root` keyword), which is always the first object of the hierarchy. Then, it uses the reference contained next to the `/Root` keyword to locate the `/Catalog` object, which is the main object outside the trailer. Each of the subsequent references is then used to reconstruct the tree. Most malicious files are based on trees that finish with objects containing suspicious actions. `PeePDF` automatically underlines them and allows dumping their stream for content analysis.

`Origami` (http://esec-lab.sogeti.com/pages/origami.html) is very similar to `PeePDF`, as it also allows us to visualize the PDF file structure. It additionally provides routines for encrypting and decrypting files, extracting metadata, and so forth.

### Code-Based Analysis

Here, the goal of the analyst is to analyze embedded scripting code without focusing on the internals of the PDF file. This analysis is usually performed to unveil the presence of scripting lines related to known vulnerabilities, which can provide clear hints on the maliciousness of the file. `PeePDF` and `Origami` both have functionalities to automatically detect suspicious strings inside JavaScript codes. However, `PhoneyPDF` is probably the best software to use in performing such analysis. In fact, this software (written in Python) first detects objects bearing JavaScript-related keywords. Then, it instruments and executes the extracted JavaScript code with a JavaScript interpreter to point out suspicious functions. Such analysis is limited by the fact that it is related only to the detection of JavaScript and ignores other attack possibilities, like SWF file embedding.

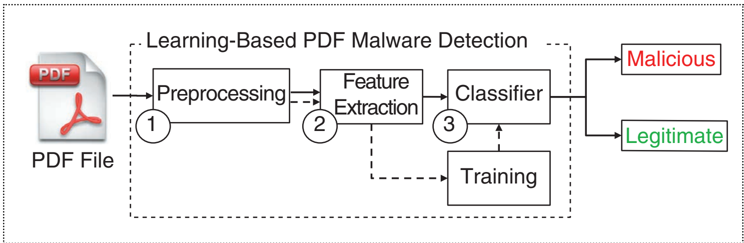


**Figure 2.** The graphical architecture of a learning-based PDF malware detection tool.

## Learning-Based PDF Malware Detection

The previously discussed forensic techniques can be used after the identification of a set of suspicious PDF files to identify the malware code responsible for the infection and characterize its behavior. The learning-based PDF malware detection tools discussed in this section have normally been proposed to prevent novel infections, but they can also be used in a forensic investigation, to identify the suspicious PDF files that demand a subsequent detailed analysis. Notably, machine learning has been increasingly applied as a key component in recent PDF malware detectors to counter the growing variability and sophistication exhibited by current PDF malware. The design of such tools is based on the three main steps, shown in Figure 2 and described in the following.

### Preprocessing

As with many other malware detection tools, the first step of PDF malware detectors is to analyze PDF files statically and/or dynamically. In the former case, the file is not executed, and information is extracted solely based on static code inspection (typically, through parsing the code). In the latter case, suspicious PDF files are dynamically executed through sandboxing in protected virtual environments, and their behavior is monitored. Dynamic analysis is usually more effective at detecting malicious files, especially when the embedded malicious code has been obfuscated to compromise static analysis. However, dynamic analysis is normally very computationally demanding in terms of both space and time resources, and it may be evaded by other techniques, like a delayed execution of the malicious exploitation code. In the following, we provide an overview of the tools and libraries typically used to extract data from PDF files in current PDF malware detection systems.

**Preprocessing with third-party software.** PDF malware detectors based on dynamic analysis normally use sandboxing or code instrumentation (e.g., `JSand` or `PhoneyPDF` [6,8]).

Conversely, detection systems based on static analysis have adopted a variety of solutions over the years.

Slayer relies on PDFiD from PDF files.[1] Its updated version (Slayer NEO[2]), instead, uses PeePDF for a more in-depth analysis of embedded files, multiple versions, and streamed objects and Origami to perform integrity checks on the file structure and content. These analyses are useful to detect PDF malware hidden with subtle embedding techniques, including anomalous or malformed files.

Library-based parsing relies on specific PDF libraries that can also be used by open source PDF readers. The most popular example is Poppler, a comprehensive PDF library that is adopted by the open source reader XPDF. PJscan[7] and Hidost[3] use Poppler to detect PDF files embedding malicious JavaScript code. Although these libraries correctly implement most of the Adobe PDF specifications, they may be vulnerable to well-crafted malformations of PDF files.

**Custom preprocessing.** We refer here to preprocessing analyses, which do not leverage any third-party PDF-specific tool or library for custom preprocessing. Typically, such processing consists of implementing a static custom parser to preprocess the input PDF files. This choice has the advantage of avoiding potential vulnerabilities of existing libraries, e.g., if they do not correctly handle some malformed files. PDFRate is a good example of a PDF malware detector exploiting a custom parsing mechanism.[4,5]

However, custom parsing itself may introduce other vulnerabilities if it does not properly follow the Adobe PDF specifications. For example, Adobe Reader completely ignores any object not referenced by the x-ref table in a PDF file. Conversely, PDFRate parses those objects. This misbehavior has been exploited[9] to evade PDFRate, through injection of well-crafted objects into PDF malware files. Because these objects are ignored by the reader, they would not compromise the malicious functionality of the embedded exploitation code but still enable evasion of the detection system. We refer interested readers to Curtis et al.[13] for an in-depth evaluation of the vulnerabilities of PDF parsing tools.

### Feature Extraction

To classify PDF files as legitimate or malicious using a learning-based algorithm, a preliminary, required step is to represent each file as a numerical vector of fixed size. This process is usually referred to as *feature extraction*.

**JavaScript-based features**. The vast majority of PDF malware relies on the embedding of malicious JavaScript code. For this reason, specific features have been exploited to detect evidence of such behavior. The detection approach PJScan[7] aims to detect the presence of malicious (obfuscated) JavaScript code by considering occurrences of suspicious application programming interface (API) calls, such as eval or replace, and of string-chaining operators such as +, among others. Lux0R[8] leverages code instrumentation to detect the presence of API calls in JavaScript code that are specifically used for PDF-related operations. Wepawet dynamically executes the embedded JavaScript code using JSand and then extracts features mostly related to method calls and shellcode memory allocation.

**Structural Features.** Structural features relate only to the characteristics of the name objects present in the PDF file. They do not consider any analysis of the embedded exploitation code. This has the advantage of being sufficiently general to detect PDF malware embedding different malicious code (e.g., JavaScript or ActionScript). However, because the malicious code is not analyzed at all, it is likely that such features can be easily misled by constructing PDF files with objects similar to those typically appearing in legitimate files. PDF malware detectors based on such features include Slayer and Slayer NEO,[1,2] Hidost,[3] and PDFRate.[4,5]

### Learning and Classification

Independent of the chosen feature representation, after feature extraction, each PDF file is represented in terms of a numerical vector $\mathbf{x} \in \mathbb{R}^d$. This abstraction allows the use of any kind of learning algorithm to perform classification of PDF documents, as described in the following.

First, a learning algorithm is trained to recognize a set of known examples $\mathcal{D} \in \{\mathbf{x}_i, y_i\}_{i=1}^n$, labeled either as legitimate $(y = -1)$ or as malicious $(y = +1)$. 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 $f(x) \in \{-1, +1\}$ that can be used to classify never-before-seen PDF files as legitimate or malicious. Clearly, the selection of an appropriate learning algorithm depends on the given data and on the feature representation. Accordingly, one normally tests different algorithms and retains the one that best fits the given application requirements. The PDF malware detectors mentioned throughout this article adopt different learning algorithms; e.g., Wepawet[6] uses a Bayesian classifier and PJScan[7] uses support vector machines, while several other approaches use classifier ensembles, including random forests and Adaboost,[1–4,8] to improve resilience against some kinds of attack.[5] We refer the reader to Table 1 for a concise overview of the main characteristics of current PDF malware detection systems.

## Evading Learning-Based PDF Malware Detection

Learning-based PDF malware detection has been shown to be effective in detecting malware samples in

**Table 1. An overview of the main characteristics of current PDF malware detectors.**

| Detector | Preprocessing | | Features | Classifier |
|---|---|---|---|---|
| Wepawet[6] | Dynamic | JSand | JS-based | Bayesian |
| PJScan[7] | Static | Poppler | JS-based | Support vector machine |
| Hidost[3] | Static | Poppler | Structural | Random forests |
| Lux0R[8] | Static | PhoneyPDF | JS-based | Random forests |
| Slayer[1] | Static | PDFID | Structural | Random forests |
| Slayer NEO[2] | Static | PeePDF and Origami | Structural | Adaboost |
| PDFRate[4] | Static | Custom | Structural | Random forests |
| PDFRate (updated)[5] | Static | Custom | Structural | Classifier ensemble |

the wild. However, it is natural to expect that the level of sophistication of the next generation of attacks will increase again, exploiting vulnerabilities of the architectural components of the detection system depicted in Figure 2 (including the learning algorithm), as envisioned in other work[9,10] In a typical evasion setting, the attacker's goal is to evade classifier detection by manipulating malware under the constraint that it preserves its intrusive functionality, according to a given level of knowledge of the targeted system. In general, the attacker may know, partially or completely, some of the training data used to learn the classification function, how features are computed from PDF files, and which learning algorithm is used.

Different attacks against PDF malware detectors have been recently proposed.[4,8,9,11,12,14] In terms of the attacker's capability, they consider only the injection of different kinds of content into a PDF malware sample. Removing objects is typically avoided to keep the functionality of the exploitation code intact. In terms of the attacker's knowledge, mimicry[4,8] and reverse mimicry[12] attacks do not exploit any specific knowledge of the attacked system. In both cases, the content of a benign file is embedded into a malicious PDF, or vice versa. In particular, in a mimicry attack, the attacker injects benign content (i.e., content extracted from one or more benign PDF files) in a malicious file to increase the probability of evading detection. Conversely, in a reverse mimicry attack, the malicious content is injected into a benign file. More sophisticated attacks, usually referred to as *evasion attacks*, have been proposed against learning-based PDF malware detectors.[9,11] These attacks exploit knowledge of the feature set and the classification function to minimize the number of modifications required to evade detection, while maximizing the probability of evasion. We refer interested readers to other studies[9–11,15] for further details on how to implement such attacks.

## Content Injection in PDF Files

Three different techniques can be used to inject content into a PDF file, as conceptually depicted in Figure 3:

1) injecting objects after the x-ref table, as done by Šrndić and Laskov[9] to evade PDFRate
2) using the versioning mechanism of the PDF i.e., injecting a new body, x-ref table, and trailer, as if the file were directly modified by the user (e.g., by using an external tool)
3) directly acting on the existing PDF graph, adding new objects to the file body and rearranging the x-ref table accordingly.

The first strategy is easy to implement, but it can be made ineffective by simply patching the preprocessing module of the PDF malware detector to be consistent with Adobe Reader. In fact, within this strategy, the injected content is ignored by Adobe Reader but not by the preprocessing module of PDFRate. This strategy can clearly be used only in mimicry and evasion attacks to add benign content to a malicious PDF file. The other two strategies, rather, can be used to perform reverse mimicry attacks by injecting malicious code into a benign PDF file. The second strategy is easier to implement but clearly also easier to spot, as it would suffice to correctly extract the additional versions embedded in the file and process them separately. The third strategy is more complex to implement and detect, as it seamlessly adds objects in a PDF file yielding a PDF that is essentially indistinguishable from a newly created one. It can be implemented using `Poppler` to manage and rearrange the x-ref table objects without corrupting the file. Existing objects can also be modified by adding other name objects and rearranging the x-ref table positions accordingly. Notably, it is important to ensure that the embedded content (i.e., the exploitation code) is correctly executed when the merged PDF is opened. This is not an easy task, as it requires injecting additional objects specifically for this purpose.

## Empirical Results on Detection Systems

Here, we report an empirical evaluation of PDF malware detection tools against reverse mimicry attacks, which require only a limited number of structural changes to the benign source file (with respect to other content-injection attacks). Content injection in reverse mimicry can be performed with the techniques depicted in Figure 3(b) and (c).

We consider here the injection of three different types of content: 1) a malicious JavaScript exploitation routine, 2) a malicious PDF file, and 3) a malicious executable (i.e., the Zeus trojan payload).

JavaScript embedding was performed by injecting the same malicious code in different benign files for two reasons: 1) we wanted to verify whether different PDF file structures could influence the detection of the same malware and 2) the current version of this embedding procedure only supports malicious codes contained in a single object. More advanced attacks usually involve spreading JavaScript codes within multiple objects. Therefore, it would have been unfeasible to use different malicious codes. In the PDF embedding attack, we injected one random malicious file [gathered from `VirusTotal` (http://www.virustotal.com)] into each benign file. We wrote efficient injection routines for both JavaScript and PDF embedding attacks by employing `Poppler`. `EXE` embedding was performed using Metasploit to automatically inject a malicious payload in each benign file.

Each of these malicious contents was hidden into 500 different benign PDF files (gathered from the Yahoo search engine), yielding a complete data set of 1,500 publicly available reverse mimicry attacks (https://pralab.diee.unica.it/en/pdf-reverse-mimicry/). `PJScan`, `Hidost`, and `Slayer NEO` were all trained with a data set composed of more than 20,000 malicious and benign files, respectively, collected from `VirusTotal` and the `Yahoo` search engine. For the sake of a fair comparison, we also trained `Slayer NEO` and `Hidost` with the same classification algorithm (Adaboost). Note also that `Slayer NEO` was used by employing both the algorithm described by Maiorca et al.[1] (keywords) and the one described by Maiorca et al.[2] (keywords and content-based features).

The results are reported in Figure 4. `PDFRate`, `Slayer Neo`, and `Hidost` are especially effective at detecting `EXE` embedding attacks, as they introduce specific keywords. However, they struggle at detecting JavaScript-based attacks, as PDF structures that are apparently malicious in terms of keywords can simply contain benign code. Content-based systems are more effective at detecting embeddings of JavaScript code, but they might fail under specific circumstances. `PJScan`, in particular, suffers from the presence of multiple embedded JavaScript codes; this may happen when a malicious script is embedded into a benign file that already contains JavaScript codes.

With respect to PDF embedding attacks, `Slayer NEO` is the only effective system that can detect them, as it automates the analysis of embedded files. In particular, as the system extracts and analyzes embedded files separately from their benign containers, its detection capabilities are not influenced by the presence of benign features. In conclusion, we can state that there is no unique solution for detecting all attacks. To perform a thorough digital investigation, each tool should be considered.

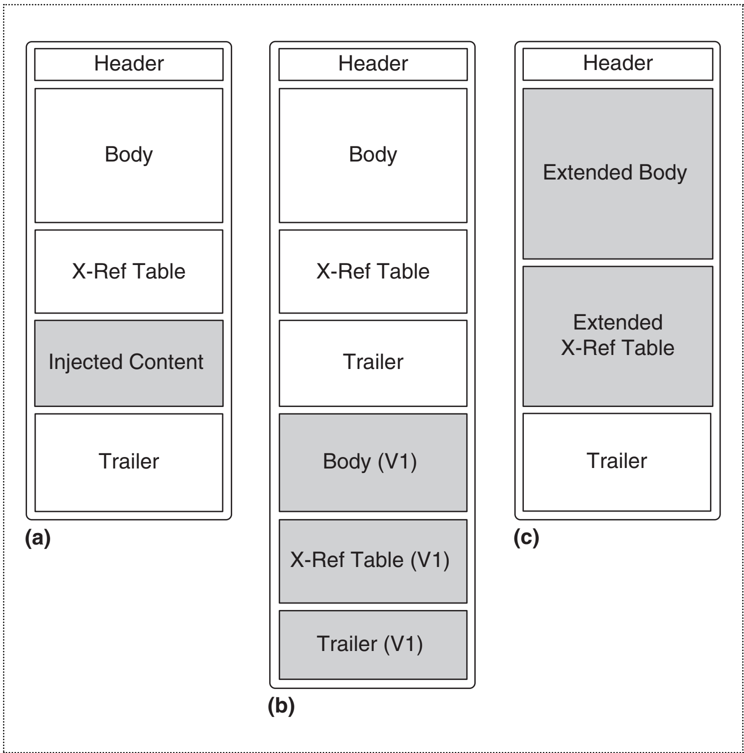


**Figure 3.** The content injection in PDF files: (a) injecting objects after the x-ref table, (b) using the versioning mechanism of the PDF, and (c) adding objects to the file body and extending the x-ref table accordingly.

In the last few decades, fueled by a flourishing underground economy, malware has grown exponentially, not only in terms of the sheer number of variants and families but also in terms of sophistication, mainly

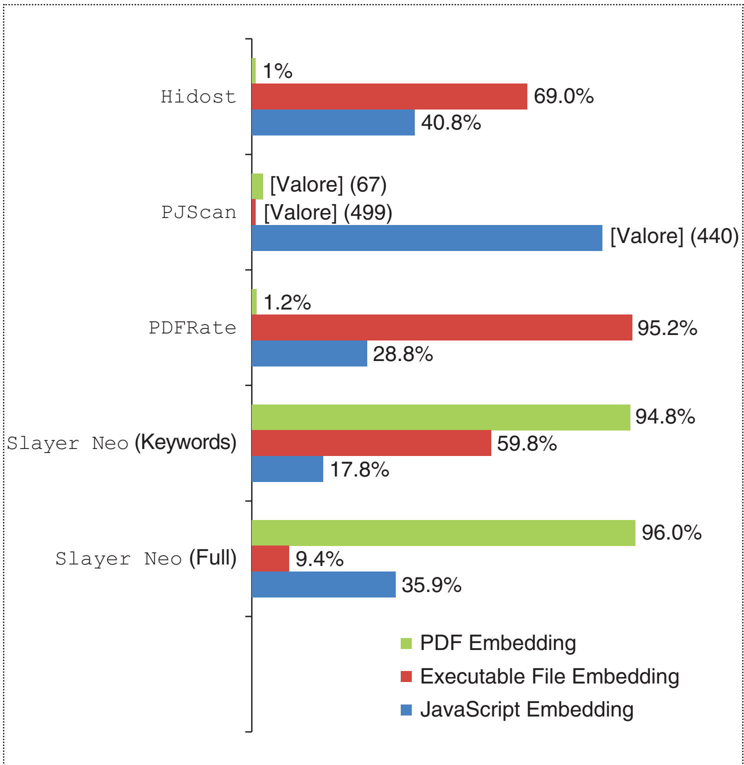


**Figure 4.** The detection rate of state-of-the-art PDF malware detectors against reverse mimicry attacks embedding different content and using 500 files per attack. Due to parsing problems, the detection rate of PJScan is estimated on a subset of files, as reported in parentheses.

to evade current detection approaches. In this article, we discussed how the PDF can be exploited by attackers to convey malware, leveraging the possibility of embedding different kinds of content and, accordingly, exploiting various potential vulnerabilities. We provided practical examples of known malware and zero-day exploits and discussed current detection systems based on machine learning. We believe that such systems can be extremely helpful for a forensic analyst to understand the suspiciousness of a PDF file and the potential root causes behind infection. Envisioning the next step of the arms race between malware and system developers, we then discussed the security properties of learning algorithms against well-crafted evasion attempts, reporting in addition some empirical results. This is an important aspect besides improving the security of other system components such as preprocessing and parsing, because machine-learning algorithms exhibit intrinsic vulnerabilities that, sooner or later, will be exploited by skilled and economically motivated attackers. From a security-by-design perspective, being proactive demands the development of adversarial learning machines, i.e., learning algorithms that explicitly account for the presence of malicious input data manipulations and provide improved security guarantees.[9,10,15] This may be one of the most relevant research challenges in the coming years. ■

## References

1. D. Maiorca, G. Giacinto, and I. Corona, "A pattern recognition system for malicious pdf files detection," in *Proc. 8th Int. Conf. Machine Learning and Data Mining Pattern Recognition*, 2012, pp. 510–524.
2. D. Maiorca, D. Ariu, I. Corona, and G. Giacinto, "A structural and content-based approach for a precise and robust detection of malicious PDF files," in *Proc. 1st Int. Conf. Information Systems Security and Privacy*, 2015, pp. 27–36.
3. N. Šrndić and P. Laskov, "Hidost: A static machine-learning-based detector of malicious files," *EURASIP J. Inf. Security*, Dec. 2016. [Online]. Available: https://jis-eurasipjournals.springeropen.com/articles/10.1186/s13635-016-0045-0
4. C. Smutz and A. Stavrou, "Malicious pdf detection using meta-data and structural features," in *Proc. 28th Annu. Computer Security Applications Conf.*, 2012, pp. 239–248.
5. C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors," in *Proc. 23rd Annu. Network & Distributed System Security Symp.*, San Diego, CA, 2016.
6. M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious Javascript code," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 281–290.
7. P. Laskov and N. Šrndić, "Static detection of malicious Javascript-bearing PDF documents," in *Proc. 27th Annu. Computer Security Applications Conf.*, 2011, pp. 373–382.
8. I. Corona, D. Maiorca, D. Ariu, and G. Giacinto, "Lux0R: Detection of malicious PDF-embedded Javascript code through discriminant analysis of API references," in *Proc. Workshop Artificial Intelligence and Security (AISEC 2014)*, pp. 47–57.
9. N. Šrndić and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *Proc. IEEE Symp. Security & Privacy, ser. SP '14*, Washington, DC, 2014, pp. 197–211.
10. B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 984–996, Apr. 2014.
11. B. Biggio et al., "Evasion attacks against machine learning at test time," in *Proc. European Conf. Machine Learning and Principles and Practice of Knowledge. Disc. Databases, ser. LNCS*, 2013, pp. 387–402.
12. D. Maiorca, I. Corona, and G. Giacinto, "Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious PDF files detection," in *Proc. 8th ACM SIGSAC Symp. Information, Computer and Communications Security*, 2013, pp. 119–130.

13. C. Curtis, X. Hu, H. Yin, A. V. Bhaskar, and M. Zhang, "Extract me if you can: Abusing PDF parsers in malware detectors," in *Proc. 23rd Annu. Network & Distributed System Security Symp.*, San Diego, CA, 2016.
14. W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers: A case study on PDF malware classifiers," in *Proc. 23rd Annu. Network & Distributed System Security Symp.*, San Diego, CA, 2016.
15. A. Kantchelian, J. D. Tygar, and A. Joseph, "Evasion and hardening of tree ensemble classifiers," in *Proc. Int. Conf. Machine Learning*, 2016, pp. 2387–2396.



**Davide Maiorca** is a postdoctoral researcher in the Department of Electrical and Electronic Engineering at the University of Cagliari, Italy. His research interests include adversarial machine learning, malware in documents and Flash applications, Android malware, and mobile fingerprinting. Maiorca received a Ph.D. in electronic engineering and computer science from the University of Cagliari in 2016. He is a Member of the IEEE and was a member of the 2016 IEEE Security & Privacy Student Program Committee. Contact him at davide.maiorca@diee.unica.it.

**Battista Biggio** is an assistant professor in the Department of Electrical and Electronic Engineering at the University of Cagliari, Italy. His research interests include secure machine learning, multiple classifier systems, kernel methods, biometrics, and computer security. Biggio received a Ph.D. in electronic engineering and computer science from the University of Cagliari in 2010. He serves as a reviewer for several international conferences and journals. He is a Senior Member of the IEEE and a member of the International Association for Pattern Recognition. Contact him at battista.biggio@diee.unica.it.





*Digital Object Identifier 10.1109/MSEC.2019.2900874*