Detection of Malicious PDF Files Based on Hierarchical Document Structure

Nedim Šrndić and Pavel Laskov
Department of Cognitive Systems
University of Tübingen
Tübingen, Germany
{nedim.srndic, pavel.laskov}@uni-tuebingen.de

Abstract

Malicious PDF files remain a real threat, in practice, to masses of computer users, even after several high-profile security incidents. In spite of a series of a security patches issued by Adobe and other vendors, many users still have vulnerable client software installed on their computers. The expressiveness of the PDF format, furthermore, enables attackers to evade detection with little effort. Apart from traditional antivirus products, which are always a step behind attackers, few methods are known that can be deployed for protection of end-user systems. In this paper, we propose a highly performant static method for detection of malicious PDF documents which, instead of analyzing JavaScript or any other content, makes use of essential differences in the structural properties of malicious and benign PDF files. We demonstrate its effectiveness on a data corpus containing about 660,000 real-world malicious and benign PDF files, both in laboratory conditions and during a 10-week operational deployment with weekly retraining. Additionally, we present the first comparative evaluation of several learning setups with regard to resistance against adversarial evasion and show that our method is reasonably resistant to sophisticated attack scenarios.

1 Introduction

Despite the recent improvements in security of the PDF rendering software, PDF documents remain a popular attack vector "for the masses". Several large-scale attacks have been recently reported using known PDF vulnerabilities [35, 15, 29]. These attacks demonstrate a surprising effectiveness of rather outdated attack vectors in the community of ordinary computer users. Additionally, several novel vulnerabilities in the Adobe Reader application have been discovered recently [13].

The existing defense mechanisms against PDF-based attacks suitable for wide-scale deployment are still inadequate for the expressive power, and hence the evasive capability of PDF malware. Even though the majority of modern antivirus products support detection of PDF-specific attacks, the detection techniques are still based on signatures and rigid heuristics. Hence they cannot quickly adapt to novel attack vectors even if the latter constitute only minor modifications of existing exploits.

Recent research on the detection of malicious PDF documents has been surprisingly sparse. Some early, format-agnostic approaches for detection of malicious file content used machine learning methods combined with byte-level n-gram analysis [21, 32]. These methods are also applicable to detection of PDF-based attacks. However, since they were developed even before the pandemic spread of PDF malware in 2009, they could not be evaluated on large corpora of modern malicious PDF documents. The majority of recent detection methods are based on dynamic analysis. Among the most popular systems of this kind are WEPAWET[1] using the sandbox JSAND [9], MALOFFICE [11] adapted from CWSANDBOX [38], and SHELLOS [34]. In general, dynamic analysis introduces significant overhead and latency, since it inherently depends on the execution of malicious code. This has motivated methods based on static analysis [19] as well as the combination of static and dynamic analysis [37]. Both of these methods are focused on detection of malicious JavaScript content in PDF documents, which relates them to a large body of work on detection of drive-by-downloads; e.g., [27, 31, 10, 6].

In contrast to prior work, the method we present in this article is intended for static detection of malicious PDF documents *without a special consideration for JavaScript content*. Although the vast majority of PDF-based exploits do rely on JavaScript, there exist two major technical challenges that make JavaScript-based detection especially difficult. The first challenge is to locate JavaScript content in a PDF file. JavaScript may be hidden deep within the logical structure of a PDF document, even beyond the locations

[1] http://wepawet.iseclab.org/

designated in the PDF Standard [25]. Any textual content in a PDF document can be interpreted as JavaScript using the `eval()` function or its equivalents. Hence, nothing prevents an attacker from distributing chunks of JavaScript code anywhere in the text and assembling them together at runtime. The second challenge lies in the high-degree expressiveness in the JavaScript language, which provides attackers with a powerful platform for code obfuscation. Although some methods have been recently proposed for detecting of obfuscated JavaScript [17], their practical utility remains unclear.

As an alternative to JavaScript-based detection, we propose analyzing the *structural properties* of PDF documents to discriminate between malicious and benign files. Instead of looking for the specific malicious content, our method evaluates documents on the basis of side-effects from malicious content within their structure. This approach is based on the intuition that, due to the complexity of the PDF format, the logical structure of PDF files conveys a significant amount of the documents' semantics. Hence, we postulate that a comprehensive analysis of structural properties would reveal a salient structural difference between malicious and benign PDF documents. In our main contribution, we propose a novel representation for the structure of PDF documents, denoted as *structural paths*, which can be used as features for automatic processing by various data analysis methods.

Extraction of structural paths can be efficiently carried out using existing PDF parsers. Parsing the structure of a document is an order of magnitude *less complex* than the interpretation and rendering of its content. Even if the parser itself may be vulnerable, its attack surface is much narrower than that of an underlying rendering application. As the vast majority of PDF-related vulnerabilities are related to JavaScript, the probability of an attack against the parser is negligible since parsing is naturally decoupled from JavaScript interpretation. The proposed methodology for feature extraction can be deployed with any parser. This reduces the opportunity for evasion techniques exploiting the ambiguities in file processing [16]. Due to its efficiency, the proposed method can be used as a light-weight *built-in detector* within the respective PDF renderer, thus completely avoiding the parser's ambiguity.

The effectiveness of the proposed method is demonstrated on the largest-ever data corpus hitherto used in evaluation of document malware detectors. We have collected over 570,000 malicious and benign PDF documents from the popular malware portal VIRUSTOTAL and augmented this dataset with a realistic sample of 90,000 benign PDF documents indexed by Google. Our laboratory experiments show that the proposed method attains the detection rate of 99.88% at the false positive rate of 0.0001% on the malicious VIRUSTOTAL and benign Google data, more than any commercial antivirus operating at VIRUSTOTAL. The same accuracy at the false positive rate of 0.0003% was observed with benign VIRUSTOTAL data which is obviously skewed towards being suspicious. In a 10-week operational deployment with weekly retraining on 440,000 malicious and benign PDF files, the proposed method consistently outperformed the best antivirus engine deployed at VIRUSTOTAL[2] in 7 weeks, scoring even in 1 week and losing in the remaining 2 weeks, due to bursts of novel data. These results demonstrate an excellent detection power of the proposed structural path features, at least for the current PDF malware observed in the real world.

Can the situation change if attackers attempt to evade our method? Despite the fact that it is content-agnostic, the proposed method is not trivial to evade. The main difficulty for the attacker lies, again, in the complexity of the PDF format. Even if the attacker knows which features are responsible for successful detection, he cannot easily remove them, as this may break the attack functionality. Further, addition of benign content, which is always an option for the attacker, may not always lead to a successful evasion. We have experimentally evaluated several potential *feature addition* attacks and observed that some classification algorithms deployed in our system are remarkably robust against this kind of attacks (less that 0.025% success rate).

In summary, the paper offers the following contributions:

1. A novel set of features is defined which enable one to effectively capture the structural difference between benign and malicious PDF files.
2. Using the proposed structural features, a classifier of PDF documents is presented whose detection accuracy, estimated on about 220,000 PDF documents under laboratory conditions, surpasses the accuracy of 43 antivirus engines deployed at VIRUSTOTAL at the time.
3. A 10-week operational deployment of the classifier with weekly retraining on 440,000 malicious and benign PDF files demonstrated the practical applicability of the proposed method under real-world conditions.
4. The new classifier does not rely on the analysis of JavaScript code and hence is not affected from potential novel techniques for hiding JavaScript in PDF documents.
5. The throughput of the classifier is evaluated and shown to be comparable to state-of-the-art static detection techniques.
6. The robustness of the proposed method against a number of evasion techniques is experimentally verified.

[2]It should be noted that only command-line versions of antivirus engines are deployed at VIRUSTOTAL.

Presentation of our methods begins with the review of related work in Section 2, which provides the motivation for the new kind of features and detection techniques investigated in the paper. The details of the PDF format necessary for understanding of technical aspects of our method are summarized in Section 3. The system architecture and the deployed algorithms are presented in Section 4. Our data corpora, the results of experiments aimed at evaluation of the detection accuracy, and the throughput of our system are reported in Section 5. Potential evasion techniques are analyzed and experimentally verified in Section 6. We end up with the discussion of our findings in Section 7 and conclusions in Section 8.

2 Related Work

Previous work on the detection of document malware shares many common ideas with the methods for detecting drive-by-downloads. This is not surprising, since the exploitation techniques underlying both kinds of malware are the same. Existing methods can be classified, with some degree of overlap, into *dynamic* analysis methods, in which documents are opened in a specially instrumented environment, and *static* methods, in which detection is carried out without malware execution.

Several key ideas have fueled the development of dynamic analysis methods. Early work followed the emulation-based approach in which a suspicious payload was executed using abstract payload execution [36] or software emulation [1, 26]. However, software emulation does not have full coverage of the instruction set and hence can be detected and evaded. To overcome this problem and improve scalability, the recently proposed system SHELLOS uses hardware virtualization instead of emulation for controlled execution of shellcode [34]. Implemented as an operating system kernel, SHELLOS is able to effectively detect shellcode in any buffer allocated by an application. However, this effectiveness has its price. While SHELLOS performs with outstanding bandwidth in detecting network-level attacks, its application to document malware suffers from high latency (on the order of seconds). Such latency is due to the fact that detection is carried out at the level of memory buffers which must be allocated by an application before they can be analyzed.

Another group of dynamic analysis methods has focused on detection of malicious behavior during execution of JavaScript. JSAND uses 10 carefully designed heuristic features to train models of benign JavaScript and detect attacks as large deviations from these models [9]. A similar approach has been successfully applied for detection of ActionScript 3 malware [24]. CUJO is built on top of a specialized JavaScript sandbox and automatically learns models of sequences of events affecting the state of the JavaScript interpreter [31]. JavaScript-specific dynamic analysis methods improve on the performance of the methods focused on shellcode detection, bringing it in the range of hundreds of milliseconds per file, while maintaining high detection accuracy and an extremely low false positive rate.

Early static methods based on n-gram analysis [21, 32] have never been evaluated on modern PDF malware. Since they do not address some essential properties of the PDF format, such as encoding, compression and encryption, they can be easily evaded by modern PDF malware using techniques similar to those used against conventional signature-based antivirus systems. PJSCAN was the first method that demonstrated feasibility of anomaly-based static detection of PDF malware focused on JavaScript content [19]. For the sake of efficiency, the JavaScript extractor of PJSCAN only searches for locations where the presence of JavaScript is prescribed by the PDF Standard. Unfortunately, this extraction strategy can be defeated by placing JavaScript code into an arbitrary location accessible via the PDF JavaScript API and fetching it with an `eval()`-like function call. Another recently proposed system, MALWARE SLAYER [23], is based on the pattern recognition methods applied to textual keywords extracted from PDF documents using the PDFID tool. It exhibits excellent detection and false alarm rates on real PDF data but is limited to the extraction functionality of PDFID and can handle neither multiple revision numbers nor objects hidden in object streams. PDFRATE is a recent learning-based, static PDF classifier operating on simple PDF metadata and byte-level file structure evaluated on a large dataset of PDF files with excellent classification performance [33]. However, it does not extract object streams, a feature that could be used to hide features from the detector.

Another two contributions must be mentioned that combine static and dynamic analysis techniques. MDSCAN [37] employs static analysis of the PDF file in order to extract all chunks of JavaScript code that can serve as an entry point to the JavaScript execution. To this end, a special-purpose parser was developed for MDSCAN, which attempts to extract additional information from a file including objects omitted from a cross-reference table as well as potentially malformed objects. The extracted scripts are executed in a JavaScript engine which emulates the engine of Acrobat Reader. During the controlled execution, all memory buffers are checked using a tool for shellcode detection based on binary emulation (NEMU). In ZOZZLE [10], the roles of the static and the dynamic components are reversed. The dynamic part of ZOZZLE extracts parts of JavaScript from the JavaScript engine of Internet Explorer before their execution, which naturally unfolds JavaScript obfuscation. The static part of ZOZZLE uses Bayesian classification built on top of the syntactic analysis of detected JavaScript code.

The comparison of related work shows a clear trade-

off exhibited in the up-to-date static and dynamic systems for document malware detection. While dynamic systems demonstrate excellent detection accuracy and low false positive rates, these advantage come at the cost of latency, performance overhead and the need for specially instrumented environments. The new method proposed in the paper attempts to bridge this gap from the static side, by boosting the detection performance while retaining the simplicity of design and computational efficiency typical for static methods. To achieve this goal, we develop the methodology for a *comprehensive static analysis* of PDF documents using an off-the-shelf PDF parser (POPPLER). Furthermore, we pay a special attention to potential evasion strategies and experimentally evaluate the robustness of the proposed method to selected attack strategies.

Before presenting the design and the evaluation of our system, we review the main features of the PDF format and document structure which are relevant for understanding the technical aspects of our method.

3 The PDF Document Structure

Portable Document Format (PDF) is an open standard published as ISO 32000-1:2008 [25] and referred to as the *PDF Reference* hereinafter.

The syntax of PDF comprises the four main elements:

- **Objects.** These are the basic building blocks in PDF.
- **File structure.** It specifies how objects are laid out and modified in a PDF file.
- **Document structure.** It determines how objects are logically organized to represent the contents of a PDF file (text, graphics, etc.).
- **Content streams.** They provide a means for efficient storage of various parts of the document content.

There are 9 basic object types in PDF. Simple object types are **Boolean**, **Numeric**, **String** and **Null**. PDF strings have bounded length and are enclosed in parentheses '(' and ')'. The type **Name** is used as an identifier in the description of the PDF document structure. Names are introduced using the character '/' and can contain arbitrary characters except *null* (`0x00`). The aforementioned 5 object types will be referred to as *primitive* types in this paper. An **Array** is a one-dimensional ordered collection of PDF objects enclosed in square brackets, '[' and ']'. Arrays may contain PDF objects of different type, including nested arrays. A **Dictionary** is an unordered set of key-value pairs enclosed between the symbols '<<' and '>>'. The keys must be *name objects* and must be unique within a dictionary. The values may be of any PDF object type, including nested dictionaries. A **Stream** object is a PDF dictionary followed by a sequence of bytes. The bytes represent information that may be compressed or encrypted, and the associated dictionary contains information on whether and how to decode the bytes. These bytes usually contain content to be rendered, but may also contain a set of other objects[3]. Finally, an **Indirect object** is any of the previously defined objects supplied with a unique object identifier and enclosed in the keywords `obj` and `endobj`. Due to their unique identifiers, indirect objects can be referenced from other objects via *indirect references*.

The syntax of PDF objects is illustrated in a simplified exemplary PDF file shown in the left-hand side of Figure 1. It contains four indirect objects denoted by their two-part object identifiers, e.g., `1 0` for the first object, and the `obj` and `endobj` keywords. These objects are dictionaries, as they are surrounded with the symbols '<<' and '>>'. The first one is the *Catalog* dictionary, denoted by its *Type* entry which contains a PDF name with the value *Catalog*. The Catalog has 2 additional dictionary entries: *Pages* and *OpenAction*. *OpenAction* is an example of a nested dictionary. It has two entries: *S*, a PDF name indicating that this is a JavaScript action dictionary, and *JS*, a PDF string containing the actual JavaScript script to be executed: `alert('Hello!');`. *Pages* is an indirect reference to the object with the object identifier `3 0`: the Pages dictionary that immediately follows the Catalog. It has an integer, *Count*, indicating that there are 2 pages in the document, and an array *Kids* identifiable by the square brackets, with two references to Page objects. The same object types are used to build the remaining Page objects. Notice that each of the Page objects contains a backward reference to the Pages object in their *Parent* entry. Altogether, there are three references pointing to the same indirect object, `3 0`, the Pages object.

The relations between various basic objects constitute the logical, tree-like *document structure* of a PDF file, illustrated in the middle part of Figure 1. The nodes in the document structure are objects themselves, and the edges correspond to the names under which child objects reside in a parent object. For arrays, the parent-child relationship is nameless and corresponds to an integer index of individual elements. Notice that the document structure is, strictly speaking, not a tree but rather a directed, potentially cyclic graph, as indirect references may point to other objects anywhere in the document structure. The root node in the document structure is a special PDF dictionary with the mandatory *Type* entry containing the name *Catalog*. Any object of a primitive type constitutes a leaf in the document structure.

The following list shows exemplary structural paths from real-world benign PDF files:

[3]This feature has been originally intended for storing collections of small objects in a compressed form. However, it has become a popular tool for obfuscation of the document structure by attackers.

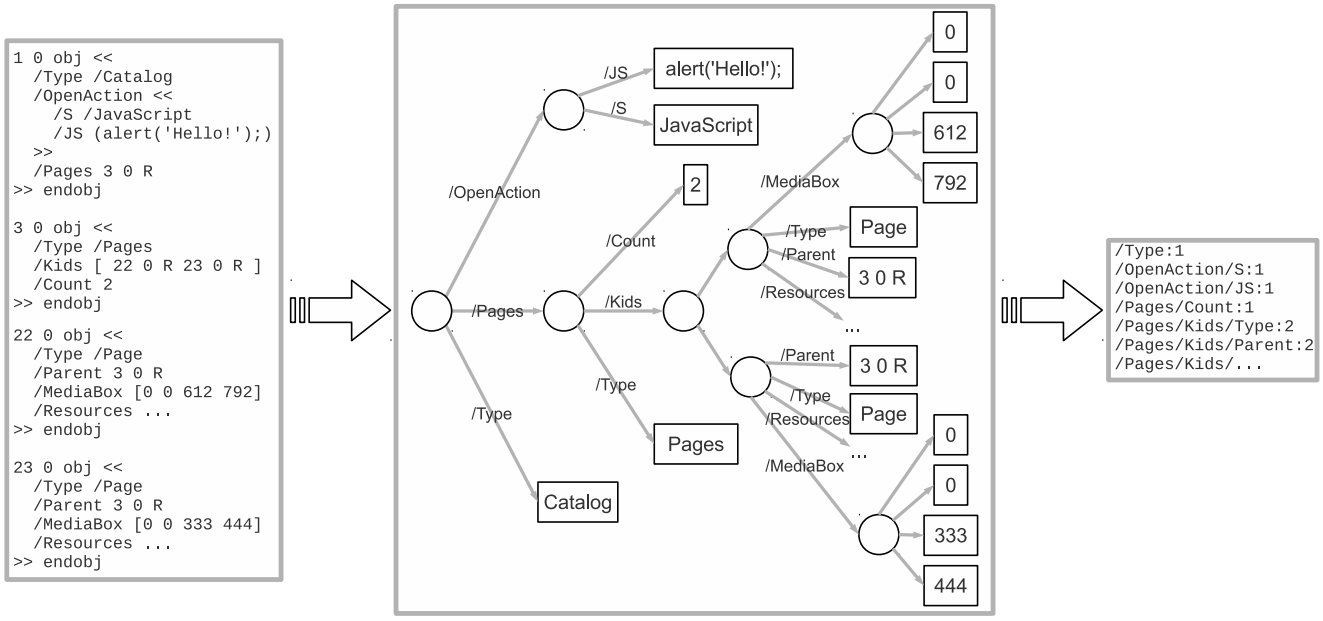

Figure 1. Various representations of the PDF structure: physical layout (left), logical structure (middle) and a set of structural paths (right)

```
/Metadata
/Type
/Pages/Kids
/OpenAction/Contents
/StructTreeRoot/RoleMap
/Pages/Kids/Contents/Length
/OpenAction/D/Resources/ProcSet
/OpenAction/D
/Pages/Count
/PageLayout
 ...
```

It was learned in experiments presented in Section 5.3.5 that these are the structural paths whose presence in a file is most indicative that the file is benign, or, alternatively, whose absence indicates that a file is malicious. For example, malicious files are not likely to contain metadata in order to minimize file size, they do not jump to a page in the document when it is opened and are not well-formed so they are missing paths such as `/Type` and `/Pages/Count`.

The following is a list of structural paths from real-world malicious PDF files, learned in the same experiment:

```
/AcroForm/XFA
/Names/JavaScript
/Names/EmbeddedFiles
/Names/JavaScript/Names
/Pages/Kids/Type
/StructTreeRoot
/OpenAction/Type
/OpenAction/S
/OpenAction/JS
/OpenAction
 ...
```

We see that malicious files tend to execute JavaScript stored within multiple different locations upon opening the document, and make use of Adobe XML Forms Architecture (XFA) forms as malicious code can also be launched from there.

In the following section, we present the general methodology and the technical instruments needed for the analysis of the document structure leading to a reliable discrimination between malicious and benign PDF documents.

4 System Design

The proposed method for structure-based detection of malicious PDF documents comprises the following two steps, schematically shown in Figure 2:

1. *Extraction of structural features.* As the basic preprocessing step, the content of a PDF document is parsed and converted into the special form, *bag-of-paths*, which characterizes the document structure in a well-defined way.

2. *Learning and classification.* The detection process is driven by examples of malicious and benign PDF doc-

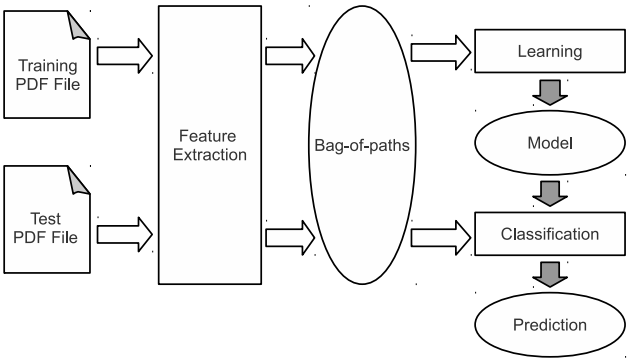

Figure 2. System architecture

uments. During the *learning* step, a model is created from the data with known labels ("training data"). The model encodes the differences between the malicious and benign data. During the *classification* step, the model is applied to new data ("test data"), to classify it as malicious or benign.

The technical realization of these two fundamental tasks is presented below.

4.1 Feature Definition

A common approach to the design of data-driven security instruments is to manually define a set of "intrinsic features" which are subsequently used for learning and classification. It was successfully applied for network intrusion detection [20, 22], botnet detection [12], detection of drive-by-downloads [9, 10, 6], and other related problems. The challenge in defining features for detection of malicious PDF documents lies in the complex structure of the PDF format. We therefore depart from the knowledge-driven strategy mentioned above and consider a richer set of potential features that capture PDFs' complexity. These features will be later automatically reduced to a smaller subset based on the available data.

The ultimate goal of the structural analysis of PDF documents is to recover all parent-child relations between its objects. The tree-like structure of PDF documents can be represented by a set of paths from the root to leaves, as shown in the rightmost part of Figure 1. Formally, we define a *structural path* to be a concatenation of the names encountered along the edges leading to a specific leaf. For notational convenience, we will use the forward slash symbol '/' as a delimiter between the names on a structural path[4]. The same structural path may occur multiple times in a document if the same path crosses some arrays and leads to different leaf objects. Empirical evidence indicates that the counts of specific paths in a document constitute a good measure of structural similarity between different documents. This motivates the choice of the set of structural paths as the intrinsic features of our system.

Due to the widespread use of indirect references in PDF documents, multiple structural paths may lead to the same object. Indirect references may even form circular dependencies, in which case the set of structural paths becomes infinite. In some semantic constructs of PDF, e.g., page trees, multiple paths are *required* in order to facilitate content rendering. Precise treatment of indirect references is only possible with directed graphs. Since the comparison of graphs is computationally difficult, we adhere to the tree-like view of the document structure and introduce additional heuristics in the following section which produce a finite set of structural paths while maintaining a reasonable semantic approximation of the existing relations.

Thus, the main operation to be performed in our feature extraction step is counting of the structural paths in a document. Additional transformations, to be referred to as "embeddings", can be applied to the path counts. The binary embedding detects the presence of non-zero counts, the frequency embedding divides the counts over the total number of paths in a document, and the count embedding refers to the path count itself. All three embeddings were experimentally evaluated and the *binary* one was chosen over the other two for its slightly better detection performance.

4.2 Extraction of PDF Document Structure

Extraction of the structural features defined in Section 4.1 must meet the following requirements:

R1: All paths must be extracted with their exact counts.

R2: The algorithm must be repeatable and robust, i.e., it must produce the same set of paths for PDF files with the same logical structure.

R3: The choice among multiple paths to a given object should be semantically the most meaningful one with respect to the PDF Reference [25].

As a first step in the extraction process, the document is parsed using the PDF parser POPPLER[5]. The key advantages of POPPLER are its robust treatment of various encodings used in PDF and the reliable extraction of objects from compressed streams. In principle, any other robust PDF parser would be suitable for extraction of structural paths, and our choice of POPPLER was only motivated by its free availability and ease of installation. The parser maintains

[4]Technically, *null* is the only character that is not allowed in a PDF name and hence, the only suitable delimiter in a structural path.

[5]`http://poppler.freedesktop.org/`, v.0.14.3.

an internal representation of the document and provides access to all fields of individual objects. Conceptually, path extraction amounts to a recursive enumeration of leafs in the document structure, starting from the *Catalog* object returned by the parser. The extracted paths are inserted into a suitable data structure, e.g., a hash table or a map, to accumulate the counts of structural paths.

Several refinements must be introduced to this general algorithm to ensure that it terminates and that the above requirements are met.

The requirement **R1** is naturally satisfied by the recursive nature of our feature extraction. Since our recursion terminates only if a leaf node is encountered, the algorithm is guaranteed to never underestimate the count of a particular path. To prevent an overestimation of the path count due to multiple paths as well as an infinite recursion due to circular references, the requirement **R3** must be enforced.

The enforcement of requirements **R2** and **R3** is tightly coupled and ultimately relies on the intelligent treatment of indirect references. Obviously, one cannot always dereference them, as this may result in an infinite recursion. One cannot also avoid their de-referencing, as the algorithm would hardly ever move beyond the root node. Hence, a consistent strategy for selective de-referencing must be implemented.

In our extraction algorithm, we approach these issues by maintaining a breadth-first search (BFS) order in the enumeration of leaf objects. This strategy assumes that the shortest path to a given leaf is semantically the most meaningful. For example, this observation intuitively holds for various cases when circular relations arise from explicit upward references by means of the *Parent* entry in a dictionary, as demonstrated by our example in Figure 1. Although we do not have further evidence to support this observation, in our experience the BFS traversal always produced meaningful paths.

Two further technical details are essential for the implementation of the BFS traversal. It is important to keep track of all objects visited at least once during the traversal and backtrack whenever an object is visited more than once. It is also necessary to sort all entries in a dictionary in some fixed order before descending to the node's children. Since no specific ordering of dictionary fields is required by the PDF Reference, such ordering must be artificially enforced in order to satisfy the requirement **R2**.

4.3 Learning and Classification

Once the counts or other embeddings over the set of structural paths are extracted, almost any learning algorithm can be applied to create a model from the given training data and use this model to classify unknown examples. For an overview of suitable algorithms, the reader may refer to any standard textbook on machine learning, e.g., [4, 14], or use any entry-level machine learning toolbox, such as SHOGUN[6] or WEKA[7]. It is beyond the scope of this paper to provide a comprehensive experimental evidence as to which machine learning method is most suitable for detection of malicious PDF documents using the structural paths. We have chosen two specific algorithms, decision trees and Support Vector Machines, for subjective reasons presented in the following section along with a high-level description of the respective method.

Although both of the chosen methods are, in principle, suitable for high-dimensional data, we have decided to artificially reduce its dimensionality for computational reasons by selecting only those sequential paths that occur in at least 1,000 files in our corpus (see Section 5.1 for a detailed description of the data used in our experimental evaluation). This reduces the number of features, i.e., structural paths, in our laboratory experiments from over 9 million to 6,087. We did not use class information for the selection of "discriminative features" as it was done, e.g., in ZOZZLE [10]. Such manual pre-selection of features introduces an artificial bias to a specific dataset and provides an attacker with an easy opportunity to evade the classifier by adding features from the opposite class to his malicious examples.

4.3.1 Decision Trees

The decision tree is a popular classification technique in which predictions are made in a sequence of single-attribute tests. Each test either assigns a certain class to an example or invokes further tests. Decision trees have arisen from the field of operational decision making and are especially attractive for security applications, as they provide a clear justification for specific decisions – a feature appreciated by security administrators. An example of a decision tree classifying whether a person may be involved in a traffic accident is shown in Figure 3.

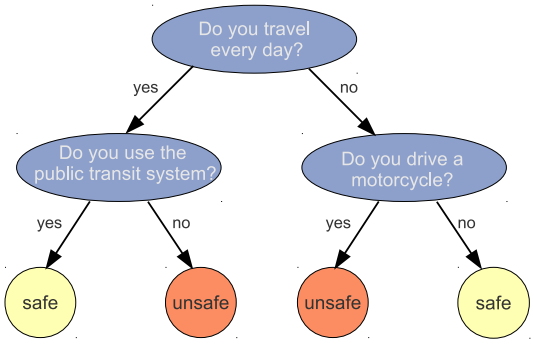

Figure 3. Example of a decision tree

[6] http://www.shogun-toolbox.org/
[7] http://www.cs.waikato.ac.nz/ml/weka/

The goal of *automatic decision tree inference* is to build a decision tree from labeled training data. Several classical algorithms exist for decision tree inference, e.g., CART [5], RIPPER [7], C4.5 [28]. We have chosen a modern decision tree inference implementation C5.0 which provides a number of useful features for practical applications, such as automatic cross-validation and class weighting[8]. It can also transform decision trees into rule sets which facilitate the visual inspection of large decision trees.

4.3.2 Support Vector Machines

The Support Vector Machine (SVM) is another popular machine learning algorithm [8]. Its main geometric idea, illustrated in Figure 4, is to fit a hyperplane to data in such a way that examples of both classes are separated with the largest possible margin M. In the case of a linear decision function, it is represented by the hyperplane's weight vector w and the threshold ρ which are directly used to assign labels y to unknown examples x:

$$y(x) = w^\top x - \rho$$

Nonlinear decision functions are also possible by applying a nonlinear transformation to input data which maps it into a feature space with special properties, the so-called *Reproducing Kernel Hilbert Space (RKHS)*. The elegance of SVM consists in the fact that such transformation can be done implicitly, by choosing an appropriate nonlinear *kernel function* $k(x_1, x_2)$ which compares two examples x_1 and x_2. The solution α to the *dual* SVM learning problem, equivalent to the primal solution w, can be used for a nonlinear decision function expressed as a comparison of an unknown example x with selected examples x_i in the training data, the so-called "support vectors" (marked with the black circles in Figure 4):

$$y(x) = \sum_{\in \in SV} \alpha_i y_i k(x, x_i) - \rho$$

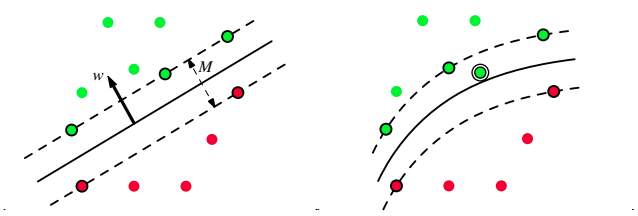

Figure 4. Linear and nonlinear SVM

Efficient implementations of SVM learning are available in various machine learning packages, including WEKA and SHOGUN. In our experiments, we used a well-known stand-alone SVM implementation LibSVM[9].

5 Experimental Evaluation

The goal of the experiments presented in this section is to measure the effectiveness and throughput of our proposed method and its operational applicability under real-world conditions. In addition, we compare the classification performance of our method to other existing methods for detecting malicious PDF files: PJSCAN and the established antivirus tools. Our evaluation is based on an extensive dataset comprising around 660,000 real-world PDF documents.

5.1 Experimental Dataset

Dataset quality is essential for the inference of meaningful models as well as for a compelling evaluation of any data-driven approach. For our evaluation, we have obtained a total of 658,763 benign and malicious PDF files (around 595 GB). Our dataset was collected from Google and VIRUSTOTAL[10], an online service that checks files uploaded by ordinary users for viruses using the majority of available antivirus programs. The collected data comprises the following 6 datasets:

D1: **VIRUSTOTAL malicious**, containing 38,207 (1.4 GB) malicious PDF files obtained from VIRUSTOTAL during the course of 18 days, between the 5th and 22nd of March 2012, which were labeled by at least 5 antiviruses as malicious,

D2: **VIRUSTOTAL malicious new**, containing 11,409 (527 MB) malicious PDF files obtained from VIRUSTOTAL 2 months later, during the course of 33 days, between the 23rd of May and 24th of June 2012, which were labeled by at least 5 antiviruses as malicious,

D3: **VIRUSTOTAL benign**, containing 79,200 (75 GB) benign PDF files obtained from VIRUSTOTAL during the course of 18 days, between the 5th and 22nd of March 2012, which were labeled by all antiviruses as benign,

D4: **Google benign**, containing 90,834 (73 GB) benign PDF files obtained from 1,000 Google searches for PDF files, without search keywords, during the course of 2,000 days, between the 5th of February 2007 and the 25th of July 2012

[8] `http://www.rulequest.com/see5-info.html`, v.2.07.

[9] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`, v.3.12.

[10] `https://www.virustotal.com/`.

D5: Operational malicious, containing 32,526 (2.7 GB) malicious PDF files obtained from VIRUSTOTAL during the course of 14 weeks, between the 16th of July and 21st of October 2012, which were labeled by at least 5 antiviruses as malicious,

D6: Operational benign, containing 407,037 (443 GB) benign PDF files obtained from VIRUSTOTAL during the course of 14 weeks, between the 16th of July and 21st of October 2012, which were labeled by all antiviruses as benign.

The VIRUSTOTAL data comprises PDF files used by people from all over the world, which brings us as close to real-world private PDF data as possible. In fact, the benign VIRUSTOTAL data is even biased towards being malicious, as users usually upload files they find suspicious. The dataset obtained by Google searches removes the bias towards maliciousness in benign data and attempts to capture the features of an average benign PDF file found on the Internet.

Note that we consider a VIRUSTOTAL file to be malicious only if it was labeled as such by at least 5 antiviruses. Files labeled malicious by 1 to 4 AVs are discarded altogether from the experiments because there is little confidence in their correct labeling, as verified empirically. Given the lack of reliable ground truth for all PDF files, we assumed the zero false positive rate for the antivirus products and could not perform their fair comparison with the proposed method with respect to the false positive rate.

5.2 Experimental Protocol

Two types of experiments were devised to evaluate the detection performance of the presented method: *laboratory* and *operational* experiments.

The three laboratory experiments operate on static data, captured in a specific point in time, where training and classification data are intermixed using *5-fold cross-validation*[11]:

- The **Standard** experiment is designed to evaluate the overall effectiveness of our method on known malicious and average benign data. To this end, we use the VIRUSTOTAL malicious dataset (**D1**) and the Google benign dataset (**D4**).
- The **Suspicious** experiment is designed to evaluate the effectiveness of our method on PDF files that ordinary users do not trust. For this experiment, we use VIRUSTOTAL malicious data (**D1**) and VIRUSTOTAL benign data (**D3**). The classification task in this experiment is harder than in the **Standard** experiment since its benign data is biased towards malicious.
- The **WithJS** experiment is designed to enable the comparison of our method to PJSCAN. For this experiment, a subset of the datasets used for the **Standard** experiment (**D1** and **D4**) was used comprising only those files that contain directly embedded JavaScript which PJSCAN can extract; i.e., 30,157 malicious and 906 benign files.

In contrast, in the two operational experiments, classification is performed on files which did not exist at all at the time of training, i.e., files obtained at a later time:

- The **Novel** experiment evaluates our method on novel malicious threats when trained on an *outdated* training set. For this experiment, we apply the models learned in the **Standard** experiment to new VIRUSTOTAL malicious data (**D2**), which is two months newer. Novel benign data was not evaluated as its observed change in this timespan was not significant.
- The **10Weeks** experiment is designed to evaluate the classification performance of our method in a real-world, day-to-day practical operational setup and compare it to the results of the best VIRUSTOTAL antivirus in the same time period. This experiment is performed on the data from the Operational benign (**D6**) and malicious (**D5**) datasets, containing files gathered during the course of 14 weeks. The experiment is run once every week, for 10 weeks starting from week 5. In every run, feature selection is performed on files gathered in the past 4 weeks. A new model is learned from scratch based on these features and data; this model is used to classify the files obtained during the current week. Thus the data obtained during weeks 1 to 4 is used to learn a model which classifies data gathered in week 5, weeks 2 to 5 are used for week 6, etc.

Note that, in practice, there are no fundamental difficulties for periodic re-training of classification models as new labeled data becomes available. The models deployed at end-user systems can be updated in a similar way to signature updates in conventional antivirus systems. As will be shown in Section 5.4, SVMs are efficient enough to allow periodic re-training of models from scratch. Our decision tree learning algorithm implementation, however, lacked the required computational performance and was not evaluated in this experiment.

[11] 5-fold cross-validation works as follows: we randomly split our data into 5 disjoint subsets, each containing one fifth of malicious and one fifth of benign files. Learning and classification are repeated five times, each time selecting a different combination of four subsets for learning and the remaining one for classification. This experimental protocol enables us to *classify every file exactly once* while ensuring that *no file processed in the classification phase was used in the learning phase for the respective model*.

5.3 Experimental Results

Both the decision tree learning algorithm and the SVM were evaluated in our laboratory experiments. For the SVM, we selected the radial basis function (RBF) kernel with $\gamma = 0.0025$ and a *cost* parameter $C = 12$, based on an empirical pre-evaluation.

5.3.1 The Standard experiment

Table 1 shows detection results for both classification algorithms in the **Standard** experiment. The top part shows the confusion matrices (the number of positive and negative files with true and false classifications) obtained by aggregating the results of all five cross-validation runs. The bottom part shows other performance indicators: the *true* and *false positive rates* and the overall *detection accuracy*.

	Decision tree	**SVM**
True Positives	38,102	38,163
False Positives	51	10
True Negatives	90,783	90,824
False Negatives	105	44
True Positive Rate	.99725	.99885
False Positive Rate	.00056	.00011
Detection Accuracy	.99879	.99958

Table 1. Aggregated results of the Standard experiment

The **Standard** experiment evaluates the overall performance of our method under laboratory conditions. As Table 1 shows, although the SVM slightly outperforms the decision tree learning algorithm, both algorithms show excellent classification performance. Very high detection accuracy (over 99.8%) was achieved, while false positives rate remained in the low promille range (less than 0.06%).

This experiment raises the question of how our method compares to modern, fully up-to-date commercial antivirus products. We were able to acquire detection results for 43 AV engines available at VIRUSTOTAL at the time of data collection. It is important to note that, because VIRUSTOTAL runs the AVs using their command-line interface, the detection capabilities of the AVs were limited to static file processing.

Figure 5 shows the comparison of true positive rates of the VIRUSTOTAL AVs and of our method using both decision trees and SVM. With the mentioned limitations in place, both of our algorithms outperform the commercial antivirus engines in this respect, and as many as 30 antivirus engines miss at least 15% of the threats.

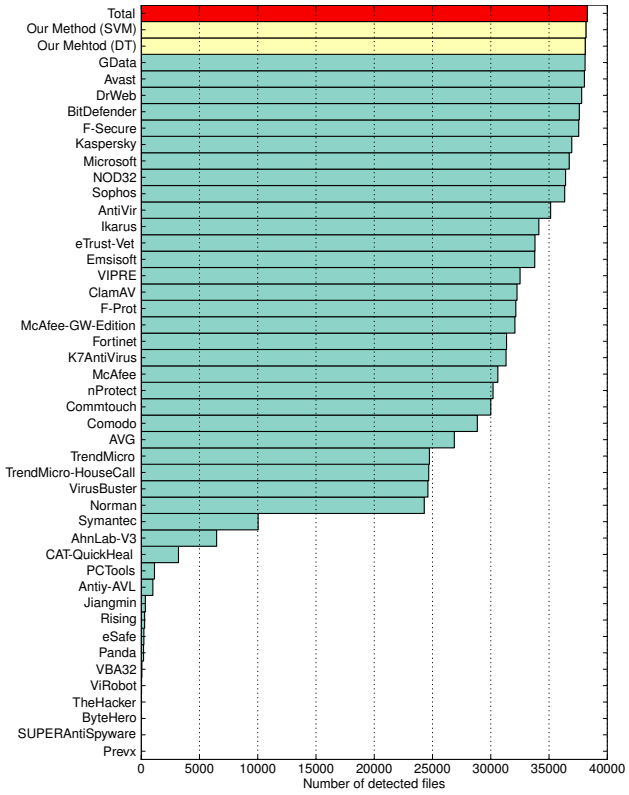

Figure 5. Comparison of our method to commercial antiviruses

5.3.2 The Suspicious experiment

Results for the **Suspicious** experiment are shown in Table 2. The classification performance of both algorithms indeed decreases when applied to this harder, suspicious data in comparison to the **Standard** experiment, however, the performance degradation is very small.

	Decision tree	**SVM**
True Positives	38,118	38,163
False Positives	68	27
True Negatives	79,132	79,173
False Negatives	89	44
True Positive Rate	.99767	.99885
False Positive Rate	.00086	.00034
Detection Accuracy	.99866	.99939

Table 2. Aggregated results of the Suspicious experiment

	Decision tree	SVM	PJScan
True Positives	30,130	30,149	21,695
False Positives	14	12	1
True Negatives	892	894	905
False Negatives	27	8	8,462
True Positive Rate	.9991	.9997	.7194
False Positive Rate	.0154	.0132	.0011
Detection Accuracy	.9986	.9993	.7275

Table 3. Aggregated results of the WithJS experiment

5.3.3 The WithJS experiment

Table 3 shows detailed experimental results for both of our algorithms and PJSCAN. Since PJSCAN performs anomaly detection; i.e., it learns using only examples of one class (malicious), the experimental protocol for PJSCAN is slightly different. Five-fold cross-validation is employed on the same data subsets as with our method, except that the benign training files are left unused for PJSCAN.

Overall, our method performs somewhat worse than in the **Standard** experiment, due to the strong class imbalance in the dataset. However, it significantly outperforms PJSCAN, a method specialized for detecting malicious JavaScript, even on a dataset carefully chosen for it. PJSCAN's high false negative rate in this experiment can be attributed to its failure to deal with intricate obfuscations. The reported effectiveness of PJSCAN is comparable to the results presented earlier with VIRUSTOTAL data [19].

The experimental results presented above represent a significant improvement in comparison to previous results on detection of PDF malware reported in the literature, summarized in Table 4 on the following page. It can be clearly seen that the detection performance of the proposed method (referred to as STRPATH) is significantly better than in previously reported results. Its closest competitor, MALWARE SLAYER [23], attains similar true positive rate but at the cost of more than 200-fold increase of the false positive rate. Both of the dynamic analysis methods, MDSCAN [37] and SHELLOS [34], generate no false positives[12] but detect only 80-90% of malware; it should be also noted that these results have been measured on an order of magnitude smaller datasets.

[12]Strangely, no data on false positive rate is reported for SHELLOS although benign data was collected. We presume it to be zero, as there is no reason to believe that a false detection of injected shellcode would have been reported on some realistic benign data.

	Decision tree	SVM
True Positives	10,681	10,870
False Negatives	728	539
True Positive Rate	.9361	.9527

Table 5. Aggregated results of the Novel experiment

5.3.4 The Novel experiment

Table 5 shows the results for the **Novel** experiment[13].

The **Novel** experiment shows that, even when our learning algorithms are trained on a 2 to 3 month-old dataset, they still achieve respectable 93% (decision tree) or 95% (SVM) true positive rates.

5.3.5 The 10Weeks experiment

Figure 6 shows the comparison of our method to the best VIRUSTOTAL antivirus[14] in the **10Weeks** experiment in terms of true positive rate (TPR). The best AV achieves an overall TPR of *92.81%*, significantly better than *87.14%* achieved by our method. However, our method consistently outperforms the best AV in 7 out of 10 weeks, and there is a draw in week 8. The performance degradation of our method in weeks 14 and, most notably, 12 has motivated a further investigation which uncovered a curious trend in VIRUSTOTAL submissions.

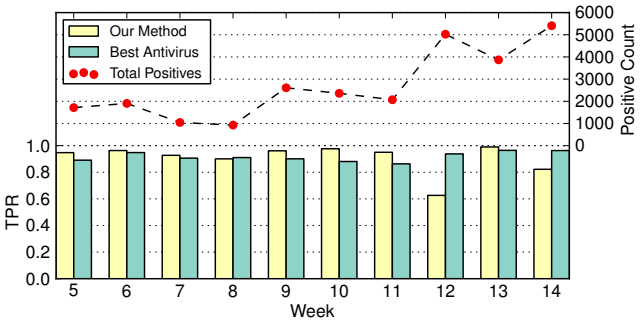

Figure 6. Comparison of the true positive rate of our method to the best antivirus for the 10Weeks experiment

The top half of Figure 6 shows the number of PDF submissions to VIRUSTOTAL detected by at least 5 AVs, i.e.,

[13]Note that some information, such as the true negative count, is missing for this experiment because it was only applied to malicious data, since changes in benign performance were negligible.

[14]The name of the best AV is not disclosed, as there are several AVs within a 5% margin of the TPR achieved in this experiment, and their rankings may change from week to week.

	STRPATH	MDSCAN	PJSCAN	SHELLOS	MALWARE SLAYER
Number of malicious samples	38,207	197	30,157	405	11,157
Number of benign samples	90,834	2,000	906	179	9,989
True positive rate	.9988	.8934	.7194	.8024	0.9955
False positive rate	.0001	0	.0011	N/A	0.0251

Table 4. Comparison of the proposed method with previous results

the number of positives per week. In the first week of October 2012, week 12, VIRUSTOTAL saw the positive submission count increase by approx. 150%, from around 2,000 to around 5,000. This elevated level of submissions has persisted to the end of the experiment. A closer inspection of the submissions in this week has revealed that there are two groups of files, one with 1,842 and the other with 2,595, which differ byte-wise from each other, but have identical PDF structure, i.e., every file in a group corresponds to the same bag-of-paths. Furthermore, there is a high similarity between the structure of the files in the two groups. The bag-of-paths of the smaller group consists of 99 structural paths, all of which are present in the other group as well. The two only differ by the presence of additional 11 structural paths in the bigger group. Files with the same bag-of-paths were also submitted later, but not before this week. This finding strongly suggests that the submissions of week 12 and later stem in great part from a single source and are generated using *fuzzing techniques*.

The cause for the false negative rate of 37% in week 12 is that all the 1,842 files in the smaller group were wrongly labeled as benign by our method. The prediction is the same for all files because they all translate into equal bags-of-paths, i.e., the same data point. A wrong classification of one data point in this case led to a false negative rate of more than $1/3$ because the test data was heavily skewed by one source producing very similar, fuzzed PDFs. In about 20 cases, these fuzzed PDFs were also missed by all the AVs.

The data point corresponding to the bag-of-paths of the smaller group of files is located on the wrong side of the SVM decision boundary, although very close to it. The addition of further 11 structural paths positioned the data point corresponding to the bag-of-paths of the larger group significantly over the decision boundary into the positive class. The reason for this lies in the fact that 8 out of 11 added structural paths are strong indicators of maliciousness in the learned SVM model[15]. In the weeks following week 12, these examples showed up in the learning stage and were correctly classified.

The performance drop in week 14 comes from a very high number of submitted files (over 900) which our parser could not open. This anomaly was not further investigated as these are either malformed PDFs which are not dangerous to the PDF renderer application but are still scanned by ignorant AVs or parser bugs, in which case it suffices to update or fix the parser or even employ a completely different one, as the method is parser-agnostic.

The overall false positive rate of our method in this experiment is 0.0655%, as in laboratory tests. The AVs do not have false positives by definition of our experiments, as the "undecided" files (the ones between 1 and 4 detections) are filtered.

5.4 Throughput

High throughput is an important consideration when dealing with large volumes of PDF data, as is the case with VIRUSTOTAL, big companies or governments. Our system was designed to handle such loads and utilize the parallel processing capabilities of modern computer systems. We have measured the time it takes to perform feature extraction, learning and classification for datasets **D1**, **D2**, **D3** and **D4** with both decision trees and SVMs. The measurements were made on a quad-core CPU with 8 GBs of RAM and a 7,200 RPM SATA hard disk with the memory caches previously cleared.

Feature extraction is the most time-consuming operation, as it requires to load all the PDF files from the hard drive. It was performed using 7 parallel processes. In total, 121 minutes and 55 seconds were spent on feature extraction for the 150 GB of data in the above mentioned datasets, of which 5 minutes and 13 seconds were spent on malicious files and 116 minutes and 42 seconds on benign files. This yields a throughput of 168 Mbit/s.

Numbers for learning and classification differ for decision trees and SVMs. They are presented in Table 6.

Since each of the 5 cross-validation runs trains on 80% of the training data, we divided the total sum of execution times for all runs by four to obtain an estimate of how long a single training would take for the entire dataset. The classification time is a simple sum of 5 individual classifications,

[15] A linear SVM was trained for the purpose of this feature interpretation which exhibits the same misclassification problem for the smaller group of files. The evaluation was performed analogue to the evasion strategy for linear SVMs presented in Section 6 by calculating and sorting weights for all features.

	Learning	Classification
Decision tree	6m 31s	52s
SVM	1m 23s	54s

Table 6. Time required for learning and classification for the Standard experiment

as each deals with 20% of the testing data. Note that executing the cross-validation runs in parallel increases performance linearly with the number of processes. Even though decision trees are significantly slower than the SVM, the overall running time is dominated by feature extraction.

The total time required for feature extraction, learning and classification using SVMs in the **Standard** experiment with the datasets **D1** and **D4** of 74.4 GB was 1 hour and 2 seconds, which yielded a total throughput of around 169 Mbit/s and a mean processing time of 28 ms per file. The high performance numbers are achieved by static detection and parallel execution. In contrast, dynamic methods such as MDSCAN (slightly less than 3,000 ms per malicious file, 1,500 ms per benign file on average) and SHELLOS (on average 5.46 seconds per file for analysis, plus additional 2 (benign) to 20 (malicious, non-ROP) seconds for buffer extraction) require orders of magnitude more time. The only other fully static method, PJSCAN, takes 23 ms per file, because it only extracts a well-defined, limited subset of the entire PDF file.

6 Evasion

An important aspect of every proposed security measure is how difficult it is to evade. Given the increasing interest to the application of learning methods for security problems, some previous work has addressed the methodology for security analysis of learning algorithms [3, 18, 2]. Following such methodology, in the following section, we present and experimentally evaluate a novel type of attacks against our method which is motivated by the specific constraints imposed by the structural features on the attacker.

The flexibility of the PDF format and the lax enforcement of its rules by the Acrobat Reader gives the attacker ample opportunity to influence both the content and the structure of the file. The only fundamental constraint on the attacker is the need to deliver the malicious content and trigger its execution by means of an appropriate attack vector. In our evasion model, we assume that the attacker has crafted a malicious PDF file that is correctly classified as malicious. The attacker's goal now is to modify the file such that it is classified as benign. We assume that the attacker *can not decrease the detection footprint by removing parts of malicious content* and is thus limited to only adding content that the classification model considers benign. Although we cannot verify that this limitation is insurmountable for an attacker, intuitively it is much easier to add arbitrary benign content to a PDF document than to change its syntactic structure of an existing document such that it is still "correctly" rendered by the viewer and triggers the exploit.

In our analysis, we assume that the attacker has complete knowledge of the detection mechanism and its classification model. Although the latter assumption may seem too restrictive, nothing prevents the attacker from collecting a surrogate dataset, similar to the one used for training, train a classifier of his own and thus obtain a good guess of the classification model. Alternatively, if our method were to be deployed on end-user systems, the files containing the learning models would be distributed to end-users, as with antivirus definitions. This would make them completely available to the attackers by means of reverse engineering. In the remaining part of this section, we analyze the potential impact of feature addition attacks on different classification algorithms deployed in our system.

In order to evade a decision tree classifier, the attacker has to modify his malicious file in such a way that will change the decision path and result in a false negative. To accomplish this, the attacker inspects the decision tree and finds the exact decision which, lead to a terminal node with the correct classification as malicious. Then the attacker must backtrack along this path and find the first non-terminal node that leads directly to a terminal node with the benign class if the test is positive. The test is positive if a certain feature exists in the file (as is the case with binary embedding) or if a feature has a count larger than a certain threshold (count embedding). In such case, it is usually straightforward for the attacker to "add" this feature into the feature vector by adding appropriate content to the PDF file. This forces the decision tree classifier into making the wrong decision. If the feature in question can not be added to this file easily (some parts of the PDF structure are strictly enforced) the attacker can continue the search for a better choice of feature. If there are no positive decisions on the path that lead directly to a benign terminal node but to a subtree instead, the attacker can still modify his file, insert such a feature, get a new decision path and repeat the search procedure. This algorithm can be easily automated. We were able to empirically verify its effectiveness by adding a previously nonexistent feature to a real-world malicious PDF sample, which forced our decision tree to misclassify it. Using random forests instead of single decision trees would make it computationally more challenging to evade detection, but they are still exposed to the same underlying problem.

A different attack strategy can be devised for the linear SVM. Recall that its decision function is computed as

$y(x) = w^\top x - \rho$. Parameters w and ρ are given by the model while the attacker controls the input vector x. The attacker's goal is to change the prediction from positive to negative. For the count embedding, this can be easily achieved by picking the dimension j with the most negative weight w_j in the model and inserting $\bar{x}_j$ content chunks corresponding to the feature j such that $y(x)$ becomes negative. Simple algebra reveals that $\bar{x}_j = (y(x) - \rho)/w_j$. We have successfully verified this evasion attack in practice. If the most negative feature cannot be inserted or can only be inserted once, the attacker can find the feature with the second most negative weight, and so on. For the binary embedding, the attacker must insert multiple features, but this is only a minor practical constraint.

Evading an SVM with an RBF kernel presents a fundamentally harder problem than in the linear case. The reason lies in the fact that the radial basis function is nonlinear, which makes the task of modifying an input vector to yield a negative result a nonconvex optimization problem. The example that we successfully used for the evasion of linear SVM did not work for RBF. As a more potent strategy, we decided to attempt evasion using a *mimicry attack*, by masquerading malicious files to look benign. In our case, this can be accomplished by copying all features of some benign file into a malicious one. To verify the efficacy of this strategy in practice, we randomly sampled 5,000 malicious and 5,000 benign training files, trained an RBF SVM and then classified another randomly selected 5,000 malicious and 5,000 benign test files. The benign file that was classified with the greatest distance from the separating hyperplane; i.e., the one classified most confidently as benign, was chosen as a camouflage in order to maximize the chance of a successful attack. We copied all the features of this file to all the 5,000 malicious feature vectors in the test set. As a result, the number of false negatives increased from 28 to 30; i.e., the strongest conceivable mimicry attack added only 2 misclassifications in 5,000 examples (0.025%)! For the linear SVM in the same setting, the mimicry attack was able to decrease detection accuracy to about 50%. This experiment practically demonstrates the strong resistance of RBF SVMs to mimicry attacks.

7 Discussion

The experimental evaluation and the analysis of potential evasion techniques presented above unequivocally demonstrate that the structure of PDF documents bears a strong discriminative power for detecting malicious documents. This finding is quite surprising given the fact that the structural difference is only a side-effect of the presence of malicious content. Although, in principle, it can be expected that attackers may find a way to camouflage malicious content within a benign structure, our experiments with several evasion strategies suggest that this task may be harder than one thinks. The most aggressive evasion strategy we could conceive was successful for only 0.025% of malicious examples tested against an off-the-shelf nonlinear SVM classifier with the RBF kernel using the binary embedding. Currently, we do not have a rigorous mathematical explanation for such a surprising robustness. Our intuition suggests that the main difficulty on attacker's part lies in the fact that the input features under his control, i.e., the structural elements of a PDF document, are only loosely related to the true features used by a classifier. The space of true features is "hidden behind" a complex nonlinear transformation which is mathematically hard to invert. This observation is corroborated by the fact that the same attack staged against the linear classifier with binary embedding had a 50% success rate; hence, the robustness of the RBF classifier must be rooted in its nonlinear transformation.

The comparison of interpretations of learned models with the success of evasion attacks leads us to the hypothesis that *explainability and robustness against evasion are antagonistic qualities of learning methods*. Indeed, the most naturally explained method, the decision tree, was the most trivial one to evade. In fact, the evasion of decision trees is so trivial that we would not recommend their use for any security-related applications. The linear SVM, whose model is relatively well interpretable (see, e.g., [30, 31] for exemplary interpretations of linear SVM in security applications), is attackable with a moderate effort. On the other hand, SVM with the RBF kernel, which exhibited the best robustness, is almost impossible to interpret. Further research effort is needed to better understand such trade-offs for specific learning methods and for development of practical compromises.

8 Conclusions

We have proposed a novel method for the detection of malicious PDF files based on the difference between the underlying structural properties of benign and malicious PDF files. By relying on structure instead of the actual content, it renders it unnecessary to deal with the very expressive PDF obfuscation techniques, interpretation of JavaScript and dynamic execution – hard problems with which related methods continue to struggle – and reaps the benefits of remaining a static method: very high throughput and robustness.

Experimental evaluation has demonstrated excellent behavior of our method in both laboratory and operational experiments on a very large dataset of around 660,000 benign and malicious PDF files. It compares favorably to recent related work from the literature (PJSCAN, MDSCAN, SHELLOS and MALWARE SLAYER). The proposed method has proved to be very effective against novel attacks, maintaining the high detection accuracy even on real PDF mal-

ware first seen more than two months after the classification model was created. A 10-week operational deployment in real-world conditions with weekly retraining has demonstrated an excellent performance of the proposed method, while also showing its difficulty to handle sudden changes in attack patterns in a timely manner. Techniques for improving its performance on strongly nonstationary data will be investigated in future work.

Computational efficiency of the proposed methods is on par with the fastest previously known static detection method (PJSCAN) and attains the throughput of 169 Mbit/s. Such performance is an order of magnitude higher than that of hybrid static/dynamic methods and almost two orders of magnitude higher than established dynamic detection methods. Thanks to the high efficiency of the proposed method, a 150 GB dataset collected from VIRUSTOTAL and Google search could be processed in two hours.

In our analysis of potential evasion strategies, we have specifically focused on the feature addition attack scenario, in which an attacker is limited in his ability to remove malicious content but is free to add benign content to avoid detection. Our analysis and experimental evaluation showed than some classifiers that were deployed in our framework, such as decision trees and linear SVM, can be easily defeated by feature addition, whereas others, such as SVM with the RBF kernel, are almost immune to sophisticated attack scenarios.

The findings presented in this paper reveal several important open issues. The surprising fact that the structure of PDF documents delivers strong indicators for the presence of malicious content calls for a deeper investigation of the syntax of the PDF format to understand the correlation of its specific features with semantics of known attacks. Such investigation should also address further potential evasion strategies against structure-based detection methods. While our experiments showed that some non-linear classification algorithms can be robust against feature addition attacks, detailed consideration of the PDF semantics is essential for understanding of content modification or feature removal attacks.

References

[1] P. Akritidis, E. Markatos, M. Polychronakis, and K. Anagnostakis. STRIDE: Polymorphic sled detection through instruction sequence analysis. In *20th International Conference on Information Security*, pages 375–392, 2005.

[2] M. Barreno, B. Nelson, A. Joseph, and J. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[3] M. Barreno, B. Nelson, R. Sears, A. Joseph, and J. Tygar. Can machine learning be secure? In *ACM Symposium on Information, Computer and Communication Security*, pages 16–25, 2006.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[5] L. Breiman, J. Friedman, J. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[6] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *International Conference on World Wide Web (WWW)*, pages 197–206, 2011.

[7] W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning (ICML)*, pages 115–123, 1995.

[8] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[9] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *International Conference on World Wide Web (WWW)*, pages 281–290, 2010.

[10] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: Fast and precise in-browser JavaScript malware detection. In *USENIX Security Symposium*, pages 33–48, 2011.

[11] M. Engelberth, C. Willems, and H. T. MalOffice – analysis of various application data files. In *Virus Bulletin International Conference*, 2009.

[12] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through IDS-driven dialog correlation. In *USENIX Security Symposium*, pages 167–182, 2007.

[13] Google warns of using adobe reader - particularly on linux. http://www.h-online.com/open/news/item/Google-warns-of-using-Adobe-Reader-particularly-on-Linux-1668153.html.

[14] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: data mining, inference and prediction*. Springer series in statistics. Springer, New York, N.Y., 2009. 2nd edition.

[15] Vorsicht bei angeblicher telekom-onlinerechnung. http://heise.de/-1545909.

[16] S. Jana and V. Shmatikov. Abusing file processing in malware detectors for fun and profit. In *IEEE Symposium on Security and Privacy*, pages 80–94, 2012.

[17] S. Kaplan, B. Livshits, B. Zorn, C. Siefert, and C. Cursinger. "nofus: Automatically detecting" + string.fromcharcode(32) + "obfuscated ".tolowercase() + "javascript code". Technical report, Microsoft Research, 2011.

[18] P. Laskov and M. Kloft. A framework for quantitative security analysis of machine learning. In *Proceedings of the 2nd ACM Workshop on AISec*, pages 1–4, Nov. 2009.

[19] P. Laskov and N. Šrndić. Static detection of malicious JavaScript-bearing PDF documents. In *Annual Computer Security Applications Conference (ACSAC)*, pages 373–382, 2011.

[20] W. Lee, S. Stolfo, and K. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[21] W.-J. Li, S. Stolfo, A. Stavrou, E. Androulaki, and A. Keromytis. A study of malcode-bearing documents. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 231–250, 2007.

[22] M. Mahoney and P. Chan. Learning rules for anomaly detection of hostile network traffic. In *International Conference on Data Mining (ICDM)*, 2003.

[23] D. Maiorca, G. Giacinto, and I. Corona. A pattern recognition system for malicious pdf files detection. pages 510–524, 2012.

[24] T. V. Overveldt, C. Kruegel, and G. Vigna. FlashDetect: ActionScript 3 malware detection. In *Recent Adances in Intrusion Detection (RAID)*, pages 274–293, 2012.

[25] PDF Reference. http://www.adobe.com/devnet/pdf/pdf_reference.html, 2008.

[26] M. Polychronakis, K. Anagnostakis, and E. Markatos. Comprehensive shellcode detection using runtime heuristics. In *Annual Computer Security Applications Conference (ACSAC)*, pages 287–296, 2010.

[27] N. Provos, P. Mavrommatis, M. Abu Rajab, and F. Monrose. All your iFRAMEs point to us. In *USENIX Security Symposium*, pages 1–16, 2008.

[28] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

[29] Blackhole crimeware kit drives web threat spike. http://www.theregister.co.uk/2012/01/26/sophos_fakeav_conficker/.

[30] K. Rieck, T. Holz, K. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behavior. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), 5th International Conference*, pages 108–125, July 2008.

[31] K. Rieck, T. Krüger, and A. Dewald. Cujo: Efficient detection and prevention of drive-by-download attacks. In *Annual Computer Security Applications Conference (ACSAC)*, pages 31–39, 2010.

[32] Z. Shafiq, S. Khayam, and M. Farooq. Embedded malware detection using markov n-grams. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 88–107, 2008.

[33] C. Smutz and A. Stavrou. Malicious PDF detection using metadata and structural features. In *Annual Computer Security Applications Conference (ACSAC)*, 2012. To appear.

[34] K. Z. Snow, S. Krishnan, F. Monrose, and N. Provos. ShellOS: Enabling fast detection and forensic analysis of code injection attacks. In *USENIX Security Symposium*, 2011.

[35] PDF malware writers keep targeting vulnerability. http://www.symantec.com/connect/blogs/pdf-malware-writers-keep-targeting-vulnerability.

[36] T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. In *Recent Adances in Intrusion Detection (RAID)*, pages 274–291, 2002.

[37] Z. Tzermias, G. Sykiotakis, M. Polychronakis, and E. Markatos. Combining static and dynamic analysis for the detection of malicious documents. In *European Workshop on System Security (EuroSec)*, 2011.

[38] C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2):32–39, 2007.