Nissim *et al. Secur Inform (2016) 5:1*
DOI 10.1186/s13388-016-0026-3

Security Informatics
a SpringerOpen Journal

RESEARCH Open Access

Keeping pace with the creation of new malicious PDF files using an active-learning based detection framework

Nir Nissim[1*], Aviad Cohen[1], Robert Moskovitch[2], Asaf Shabtai[1], Matan Edri[1], Oren BarAd[1] and Yuval Elovici[1]

Abstract

Attackers increasingly take advantage of naive users who tend to treat non-executable files casually, as if they are benign. Such users often open non-executable files although they can conceal and perform malicious operations. Existing defensive solutions currently used by organizations prevent executable files from entering organizational networks via web browsers or email messages. Therefore, recent advanced persistent threat attacks tend to leverage non-executable files such as portable document format (PDF) documents which are used daily by organizations. Machine Learning (ML) methods have recently been applied to detect malicious PDF files, however these techniques lack an essential element—they cannot be efficiently updated daily. In this study we present an active learning (AL) based framework, specifically designed to efficiently assist anti-virus vendors focus their analytical efforts aimed at acquiring novel malicious content. This focus is accomplished by identifying and acquiring both new PDF files that are most likely malicious and informative benign PDF documents. These files are used for retraining and enhancing the knowledge stores of both the detection model and anti-virus. We propose two AL based methods: exploitation and combination. Our methods are evaluated and compared to existing AL method (SVM-margin) and to random sampling for 10 days, and results indicate that on the last day of the experiment, combination outperformed all of the other methods, enriching the signature repository of the anti-virus with almost seven times more new malicious PDF files, while each day improving the detection model's capabilities further. At the same time, it dramatically reduces security experts' efforts by 75 %. Despite this significant reduction, results also indicate that our framework better detects new malicious PDF files than leading anti-virus tools commonly used by organizations for protection against malicious PDF files.

Keywords: Active learning, Machine learning, PDF, Malware

Introduction

Cyber-attacks aimed at organizations have increased since 2009, with 91 % of all organizations hit by cyber-attacks in 2013.[1] Attacks aimed at organizations usually include harmful activities such as stealing confidential information, spying and monitoring an organization, and disrupting an organization's actions. Attackers may be motivated by ideology, criminal intent, a desire for publicity, and more. The vast majority of organizations rely heavily on email for internal and external communication. Thus, email has become a very attractive platform from which to initiate cyber-attacks against organizations. Attackers often use social engineering in order to encourage recipients to press a link or open a malicious web page or attachment. According to Trend Micro,[2] attacks, particularly those against government agencies

[1] http://www.humanipo.com/news/37983/91-of-organisations-hit-by-cyber attacks-in-2013/.

[2] http://www.infosecurity-magazine.com/view/29562/91-of-apt-attacks-start-with-a-spearphishing-email/.

*Correspondence: nirni@post.bgu.ac.il
[1] Department of Information Systems Engineering, Ben-Gurion University of the Negev, Beersheba, Israel
Full list of author information is available at the end of the article

and large corporations, are largely dependent upon Spear-Phishing[3] emails.

An incident in 2014 aimed at the Israeli ministry of defense (IMOD) provides an example of a new type of targeted cyber-attack involving non-executable files attached to an email. According to media reports,[4] the attackers posed as IMOD representatives and sent email messages with a malicious portable document format (PDF) file attachment which, when opened, installed a Trojan horse enabling the attacker to control the computer.

Non-executable files attached to an email are a component of many recent cyber-attacks as well. This type of attack has grown in popularity, in part because executable files (e.g., *.EXE) attached to emails are filtered by most email servers due to the risk they pose and also because non-executables (e.g., *.PDF, *.DOC, etc.) are not filtered out and are considered safe by most users. Non-executable files are written in a format that can be read only by a program that is specifically designed for that purpose and often cannot be directly executed. For example, a PDF file can be read only by a PDF reader such as Adobe Reader or Foxit Reader. Unfortunately, non-executable files are as dangerous as executable files, since their readers can contain vulnerabilities that, when exploited, may allow an attacker to perform malicious actions on the victim's computer. F-Secure's 2008–2009 report[5] indicates that the most popular file types for targeted attacks in 2008–2009 were PDF and Microsoft Office files. Since that time, the number of attacks on Adobe Reader has grown.[6]

To prevent such attacks, defensive tools such as firewalls, Intrusion detection systems (IDSs), intrusion prevention systems (IPSs), anti-viruses, sandboxes, and others are used; however, these tools have limitations in the detection of attacks that are launched via non-executable files, particularly when a sophisticated advanced persistent threat attack is executed against an organization. These tools rely heavily on the store of known signatures maintained by anti-virus vendors and communicated to clients. Their main limitation is their inability to detect very new unknown types of attacks through known signatures, due to the time lag that exists between when a new unknown malware appears and the time anti-virus vendors update their clients with the new signature. During this period of time, many computers are vulnerable to the spread and actions of new malware [1], [2]. Thus, the currently known signatures of the clients of anti-virus vendors are consistently not up to date.

In order to effectively analyze tens of thousands of new, potentially malicious PDF files on a daily basis, anti-virus vendors have integrated a component of a detection model based on machine learning (ML) and rule-based algorithms [3] into the core of their signature repository update activities. However, these solutions are often ineffective, because their knowledge base is not adequately updated. This occurs because many new, potentially malicious PDF files are created every day, and only a limited number of security researchers are tasked with manually labeling them. Thus, the problem lies in prioritizing which PDF files should be acquired, analyzed, and labeled by a human expert.[7]

In this study we present an active learning (AL) based framework for frequently updating anti-virus software with new malicious PDF files. The framework focuses on improving anti-virus tools by labeling those informative PDF files (potentially malicious or very informative benign files) that are most likely to improve the detection model's performance and, in so doing, enrich the signature repository with as many new PDF malware files as possible, further enhancing the detection process. Specifically, the presented framework favors files that contain new content. We focus on PCs, the platform most used by organizations; mobile platforms are outside the scope of this study.

Background

Before we provide a review of existing techniques and known methods of attack, it is worthwhile to mention that Adobe Reader version X, released in 2011, offers a new feature called Protected Mode Adobe Reader (PMAR). Protected mode uses a sandbox[8] technique in order to create an isolated environment for the Acrobat Reader rendering agent to run while reading a PDF file. As a result, malicious code actions cannot affect the operating system. However, most organizations are not up-to-date with the newest versions of software, including PDF readers,[9] and thus they are exposed to many well-known attacks that exploit vulnerabilities that exist in previous versions of Adobe Reader. In addition, PDF files can be utilized for malicious purposes using a variety of techniques. In order to explain how PDF files can be exploited when created or manipulated by an attacker, we first describe the structure of a viable PDF file.

[3] http://searchsecurity.techtarget.com/definition/spear-phishing.

[4] http://www.israeldefense.co.il/?CategoryID=512&ArticleID=5766.

[5] http://www.f-secure.com/weblog/archives/00001676.html.

[6] http://www.computerworld.com/article/2517774/security0/pdf-exploits-explode–continue-climb-in-2010.html.

[7] http://www.kaspersky.com/se/images/KESB_Whitepaper_KSN_ENG_final.pdf.

[8] http://searchsecurity.techtarget.com/definition/sandbox.

[9] http://www.csoonline.com/article/2133359/malware-cybercrime/cyberattack-highlights-software-update-problem-in-large-organizations.html.

PDF file structure

A PDF is a formatting language first conceived by John Warnock, one of the founders of Adobe Systems.[10] The first version, version 1.0, was introduced in 1993, and the most current version is XI[11] (11.0.10), was released on December 9, 2014. The PDF specification is publically available,[12] and thus can be used by anyone. In addition to simple text, a PDF can include images and other multimedia elements; a PDF can be password protected, execute JavaScript, and more. Likewise, the PDF is supported in all of the prominent operating systems for the PC and mobile platforms (e.g., Microsoft Windows, Linux, MacOS, Android, Windows Phone, and iOS).

A PDF file is a set of interconnected objects built hierarchically. The PDF file structure is depicted in Fig. 1 and is comprised of four basic parts according to the Adobe PDF Reference[13] [4], [5], and [6]:

1. **Objects**—basic elements in a PDF file:
   - **Indirect objects**—objects referenced by a number
   - **Direct objects**—objects that are not referenced by a number
   - **Object types**:
     - **Boolean**—for true or false values
     - **Numeric**:
       Integer value
       Real value
     - **String**:
       Literal string—a sequence of literal characters enclosed with ( ) brackets
       Hexadecimal string—a sequence of hexadecimal numbers enclosed with < > brackets
     - **Name**—a sequence of 8-bit characters starting with /
     - **Null**—an empty object represented by the keyword 'null'
     - **Array**—an ordered sequence of objects enclosed with [] brackets that can be composed of any combination of object types, including another array
     - **Dictionary**—an unordered sequence of key-value pairs: keys being names which should be unique in the dictionary, and values being any object type. Most of the indirect objects in a PDF document are dictionaries, and dictionaries are enclosed with ≪ ≫ brackets.
     - **Stream**—a special dictionary object followed by a sequence of bytes enclosed with the keywords "stream" and "endstream." The information inside the stream may be compressed or encrypted by a filter. The prefix dictionary contains information on whether and how to decode the stream. Streams are used to store data such as images, text, script code, etc.
2. **File structure**—defines how the objects are accessed and how they are updated. A valid PDF file consists of the following four parts:
   1. **Header**—the first line of a PDF file which specifies the version number of PDF specification which the document uses. Header format is " % PDF-*[version number]*."
   2. **Body**—the largest section of the file which contains all the PDF objects. The body is used to hold all of the document's data that is shown to the user.

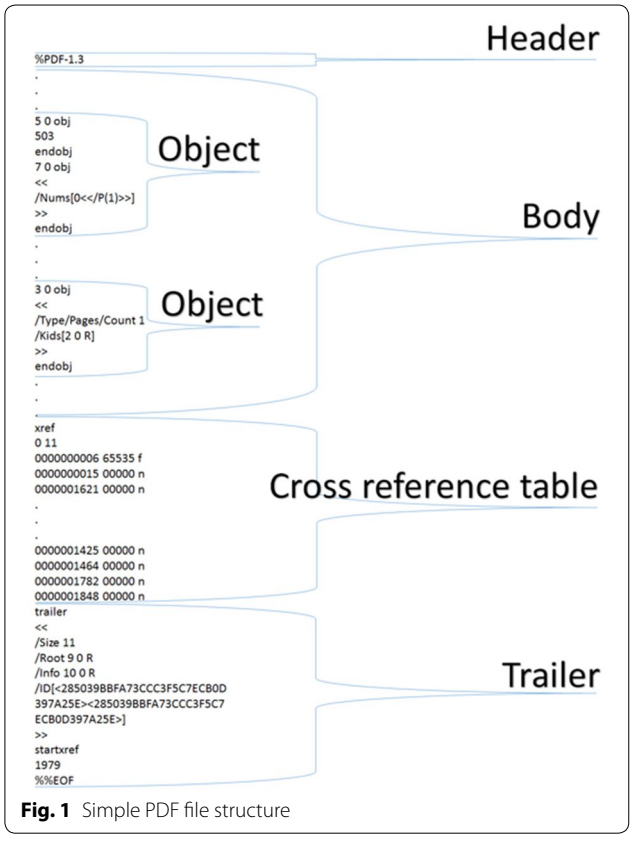

Fig. 1 Simple PDF file structure

[10] http://partners.adobe.com/public/developer/tips/topic_tip31.html.

[11] http://get.adobe.com/reader/.

[12] http://www.adobe.com/devnet/pdf/pdf_reference.html.

[13] http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf.

3. **Cross reference**—a table that includes the position of every indirect object in the memory and allows random access to objects in the file, so the application does not need to read the entire file to locate a particular object. Each object is represented by one entry in the table which is always 20 bytes long.
4. **Trailer**—provides relevant information about how the application reading the file should find the cross reference table and other special objects. The trailer also contains information about the number of revisions made to the document. All PDF readers should begin reading a file from this section.

3. **Document structure**—defines how objects are logically and hierarchically organized to reflect the content of a PDF file. Each page in the document is represented by a page object which is a dictionary that includes references to the page's content. The root object of the hierarchy is the catalog object which is also a dictionary.
4. **Content streams**—objects that contain instructions which define the appearance of the page.

Techniques and possible attacks via PDF files

JavaScript code attacks

PDF files can contain client-side JavaScript code for legitimate purposes including: 3D content, form validation, and calculations. JavaScript code can reside on a local host or remote server, and can be retrieved using/URI or/GoTo commands [7]. The main indicator for JavaScript code embedded in a PDF file is the presence of the/JS keyword in some dictionaries (as previously explained in the previous subsection II-A). However, an object containing such a dictionary can be placed in a filtered stream. The/JS keyword will not be visible in plain text and therefore may obstruct detection techniques that rely on these keywords [8]. The primary goal of the malicious JavaScript code inside a PDF file is to exploit a vulnerability in the PDF viewer in order to divert the normal execution flow to the embedded malicious JavaScript code. This can be achieved by performing a heap spraying[14] or buffer overflow attack implemented through JavaScript. Another malicious activity that can be carried out using JavaScript is downloading an executable file from the Internet which initiates an attack on the victim's machine once executed. Alternatively, JavaScript code can also open a malicious website that can perform a variety of malicious operations targeting the victim's machine. According to [9], most attacks related to PDF files are conducted using JavaScript code embedded inside a PDF.

JavaScript code obfuscation is legitimately used to prevent reverse engineering of proprietary applications. However, it is also used by attackers to conceal malicious JavaScript code and prevent it from being recognized by signature based detectors or detectors based on lexical analysis of code (such as [7]), and to reduce readability by a human security analyst.

Filters are used in PDFs to compress data in order to enhance compactness or facilitate encoding. By using filters, an attacker can conceal the malicious JavaScript code, rendering it undetectable or unreadable. Multiple filters can be applied to a stream, one after the other. The filter names used must be declared in the stream dictionary. Available filters and their primary purposes are discussed by P. Baccas and J. Kittilsen [10], [11]. Table 1 summarizes various code obfuscation techniques employed by attackers [4].

Embedded file attack

A PDF file can contain other file types within it, including HTML, JavaScript, SWF, XLSX, EXE, Microsoft Office files, or even another PDF file. An attacker can use this functionality in order to embed a malicious file inside a benign file. This way, the attacker can leverage the vulnerabilities of other file types in order to perform malicious activity, such as exploiting the vulnerability of a Flash file embedded within a PDF file. CVE-2010-3654 is an example of arbitrary code execution that can be achieved by embedding a specially crafted Flash movie file (.SWF) into a PDF file. The embedded file can be opened when the PDF file is opened using embedded JavaScript code or by other techniques such as PDF commands (e.g.,*/Launch* or*/Action/EmbeddedFiles*) which are usually combined with sophisticated social engineering techniques.[15] Usually, embedded malicious files are obfuscated in order to avoid detection. Adobe Reader PDF viewer versions 9.3.3 and above restrict file formats that can be opened and do not allow the launching of an embedded executable file such as *.EXE, *.BAT, or *.CMD because of its blacklist which is based on file extension. However, Python code (*.PY) is not blacklisted and can perform malicious activities when executed.[16]

Form submission and URI attacks

In 2013, Valentin Hamon [12] presented practical techniques that can be used by attackers to execute malicious code from a PDF file. Adobe Reader supports the option

[14] *Heap Spraying* A technique used in exploits to assist random code execution.

[15] http://www.zdnet.com/article/hacker-finds-a-way-to-exploit-pdf-files-without-a-vulnerability/.

[16] http://www.decalage.info/file_formats_security/pdf.

Table 1 Code obfuscation techniques in PDF files that can be used by an attacker

Obfuscation technique	Details
Separating malicious code over multiple objects	Malicious code is spread among multiple objects. Code chunks are collected and merged and compiled to form a malicious piece of code only during runtime. This makes it difficult for static analysis detectors to recognize the malicious code
Applying filters	Filters are used to conceal malicious code
White space randomization	Random white spaces are inserted in the malicious code in order to evade recognition by signature based maliciousness detectors. White spaces do not affect the code since JavaScript ignores them
Comment randomization	Random comments are inserted in the malicious code in order to evade recognition by signature based maliciousness detectors. Comments do not affect the code since JavaScript ignores them
Variable name randomization	Changing the variable's name randomly in order to fool signature based maliciousness detectors
Integer obfuscation	Representing numbers in a different way which can be used to hide a specific memory address
String obfuscation	Making changes to string in order to make it difficult for a human analyst to understand the code (e.g., by splitting string into several substrings)
Function name obfuscation	Hiding the name of the function used which can provide a clue about the code's intention. This is done by creating a pointer with a random name, pointing to the required function
Advanced code obfuscation	String can hold encrypted malicious code. The decryption process takes place during runtime, just before usage. Metadata fields and even the document's words can also be used to store malicious code
Block randomization	Changing the syntax of the code but not its action
Dead code	Inserting blocks of code that are not intended to be executed
Pointless code	Inserting blocks of code that do not perform anything

of submitting the PDF form from a client to a specific server using the*/SubmitForm* command. Several file formats can be used for that purpose, one of which is the forms data format (FDF), the default format based on XML. Adobe generates an FDF file from a PDF in order to send the data to a specified URL. If the URL belongs to a remote web server, it is able to respond. Responses are temporarily stored in the %APPData % directory which automatically pops up in the default web browser. An attack can be performed by a simple request to a malicious website that will automatically pop up on the web browser, and the malicious website can exploit a vulnerability in the user's web browser. The author also showed that security mechanisms such as the protected mode of Adobe Reader X or the URL Security Zone Manager of the Internet Explorer web browser can be disabled easily by changing the corresponding registry key, a change that can be implemented with user privileges. However it should be noted that vulnerabilities[17] that target Adobe Reader X and XI have been released[18] and are being exploited as well. Moreover, a URI[19] address can be used (instead of a URL) to refer to any file type located remotely (both executable and non-executable files, including *.EXE and *.PDF). One interesting scenario involves launching a malicious PDF file from a benign one. Other attacks described in the paper include an attack in which a benign PDF simply submits its form to the attacker's PHP web server. The server searches the submitted form header for the Adobe Reader version using regular expressions. Once the server identifies the user's Adobe Reader version, it sends back a malicious PDF that exploits a vulnerability corresponding to that version. The returned PDF is automatically launched. Another attack described is the Big Brother attack. When the user opens a PDF, it automatically downloads a malicious executable file using the web browser (URI address). This attack can be performed prior to the previously described form submission attack, in order to make changes to registry keys that configure the security mechanisms discussed above.

Related work

Existing anti-virus software is not adequately effective against unknown non-executable malicious PDF files [13]. Advanced methods for the detection of new or unknown malicious PDF files are based primarily on classifiers induced by ML algorithms. These methods leverage information from features extracted from labeled PDF files using static or dynamic analysis. Static analysis methods examine and evaluate a suspicious file solely by analyzing its code, without actually executing it. Alternatively, dynamic analysis methods execute the file, usually in an isolated environment (sandbox), and examine its actions and behavior during runtime.

[17] http://www.cvedetails.com.

[18] https://blogs.mcafee.com/mcafee-labs/analyzing-the-first-rop-only-sandbox-escaping-pdf-exploit.

[19] *URI* According to RFC2396, URI is a compact string of characters used for identifying an abstract or physical resource. It is an extension of URL used for identifying any web resource (not limited to web pages).

In recent years, the need to enhance security in the face of attacks based on malicious PDF files has led to increased research in this area. Most of the academic work focuses on static analysis, since it requires less computational resources and is much faster than dynamic analysis. Static analysis methods usually examine and inspect the document's embedded JavaScript code, file structure, or metadata (such as the number of specific streams, objects, keywords, etc.). However, static analysis has drawbacks as well, including the inability to detect well obfuscated code that acts maliciously during runtime, in contrast to dynamic analysis that will likely detect that code. Here we present the most relevant studies, however a more comprehensive analysis of related work can be found in our recent survey study [4].

Srndic and Laskov [7] introduced *PJScan*,[20] a static analysis and anomaly detection tool for the detection of malicious JavaScript code inside a PDF file. After the JavaScript code has been found and extracted, a lexical analysis is applied using a JavaScript interpreter. Lexical analysis represents the JavaScript code as a sequence of tokens. For example, left parenthesis, plus, right parenthesis, etc. By using these tokens *PJScan* tries to induce learning detection models that differentiate between benign and malicious PDF files. Liu et al. [14] combined both static and dynamic analysis to detect potential infection attempts in the context of JavaScript execution. First, they extract a set of static features, and then they insert context monitoring code into a PDF document, a code that later cooperates with the runtime monitor used for the detection task. Additional work done by Corona et al. [15] in which they presented the Luxor system which applied this combination of static and dynamic analysis as well. Their idea involved translating the JavaScript code into an API reference pattern, and accumulating the times of presences for every API reference. By doing this they tried to find a discriminative set of references that characterizes malware code in order to automatically differentiate this code from benign code within PDF files.

Yatamanu et al. [9] introduced two different static methods for clustering PDF files based on tokenization of their embedded JavaScript. The article focuses on establishing a clustering method for the identification of similar scripts that have been obfuscated using different techniques. For each examined PDF file, a fingerprint is created, consisting of a set of unique JavaScript tokens and their frequencies.

Maiorka et al. [8] introduced the PDF Malware Slayer (*PDFMS*), a static analysis tool which characterizes PDF files according to the set of embedded keywords and their frequencies. The files are characterized by the presence of specific chosen keywords, for example:/JS,/JavaScript,/Encrypt, obj, stream, filter, etc. The main contribution is the ability to detect malicious PDF files whether or not they contain JavaScript code, unlike previously described tools which detect malicious files only if they contain JavaScript code (e.g., *PJScan* [7]). However, an attacker can learn which keywords characterize benign files and inject these keywords inside a malicious file in order to bypass *PDFMS*, thus demonstrating the tool's weakness.

Smutz and Stavrou [16] presented *PDFRate*, a framework for the detection of malicious PDF files which is based on meta-features extracted from a document's content. The extracted features include the number of font objects, average length of stream objects, and the number of lower case characters in the title. In total, 202 features were chosen for classification.

Pareek and Eswari [17] introduced two different static analysis methods that do not rely upon a PDF parser to extract data from the file. Alternatively, the methods apply the analysis on the whole file, after its content is converted to hexadecimal dumps or byte sequences. The first method is based on entropy and is used to measure the uncertainty or randomness in a given dataset, assuming the level of uncertainty of a malicious file should be less than that of a benign file with a similar format. Low entropy in a file is not a strong indicator of maliciousness, however it can be a useful detection feature in combination with other features. The second method leverages the n-gram approach [18] in order to distinguish between benign and malicious PDF files.

Srndic and Laskov [6] introduced a method that makes use of essential differences in the structural properties of malicious and benign PDF files. Their static method evaluates documents based on side effects of malicious content within their structure by evaluating their structural paths and the frequencies of these paths. This detection model was found to be effective at differentiating between malicious and benign PDF files and was effective against new unknown malicious files that were created 2 months after the classification model was built.

The following dynamic analysis methods focus on the analysis of embedded JavaScript code (which may or may not reside in a PDF file) during runtime. All of these methods include a dynamic step, either in the feature extraction or analysis phase. *MDScan* [13] and *PDF Scrutinizer* [19] start with a static extraction of the embedded JavaScript code from a PDF file and then execute the extracted code using a JavaScript engine. The extracted JavaScript code is examined during runtime using

[20] The source code of PJScan and its underlying library (libPDFJS) can be found at http://sourceforge.net/p/pjscan/home/Home/.

heuristics in order to detect suspicious or malicious activity. Snow et al. [20] presented *ShellOS*, a framework for the detection of code injection attacks based on code analysis during runtime. Lu et al. [21] introduced *MPScan*, a technique that integrates static malware detection and dynamic JavaScript de-obfuscation. *MPScan* hooks[21] the Adobe Reader's native JavaScript engine; thus embedded codes (JavaScript source and operational code) can be extracted during execution and evaluated by the static detection module.

While the presented dynamic analysis methods execute the JavaScript code in order to detect malicious behavior, they differ in the way that they extract the JavaScript code. Generally speaking, the more exhaustive the extraction of JavaScript code is, the better the dynamic analysis can be in terms of detection ability. Nevertheless, an attempt to extract JavaScript code from a file statically may fail and result in the extraction of JavaScript code that does not accurately represent the behavior of the file. The reasons may be varied; for example, this could be due to cases in which the code can be well obfuscated or located in irregular locations within the PDF file. Dynamic extraction is more robust compared to static extraction in these areas.

It is important to clarify that the academic solutions in this category do not perform a dynamic analysis on the entire file, rather dynamic analysis is only performed on the JavaScript code that was extracted from the PDF file. This is in contrast to some commercial solutions that execute the PDF file and examine its behavior and influence on the host operating system during runtime (full dynamic analysis). Full dynamic analysis consumes significantly more resources than the dynamic analysis found in academic solutions but is probably better at detecting malicious PDF files, because it provides a better indication of the file's real purpose. Moreover, the full dynamic analysis approach is robust against code obfuscation, since it does not analyze string code or pretend to extract the code from the file. This approach is most like the examination of suspicious code by a security expert with forensic tools. Wepawet [22], an example of a well-known and widely used full-dynamic analysis tool, was presented in 2010 by Cova et al. In-Nimbo Sandboxing system for PDF files, presented by Maas et al. [23], is based on conducting vulnerable or malicious computations on a virtual machine instances in a remote cloud computing environment, and by so doing, preventing the ability of malware to affect the host system.

Studies in several domains have successfully applied AL in order to improve the efficiency of labeling examples needed to maintain the performance of an ML classifier. Unlike random (or passive) learning, in which a classifier randomly selects examples from which to learn, the ML based classifier actively indicates the specific examples which are commonly the most informative examples for the training task and should be labeled. SVM-Simple-Margin [24] is a current AL method considered in our experiments. Active learning was successfully used to enhance the detection of unknown computer worms [25] and malicious executable files targeting the Windows OS [26]. Such methods are used in the current study in order to enhance the detection of malicious PDF files.

Suggested framework and methods

A framework for improving detection capabilities

Figure 2 illustrates the framework and the process of detecting and acquiring new malicious PDF files by maintaining the updatability of the anti-virus and detection model. In order to maximize the suggested framework's contribution, it should be deployed in strategic nodes (such as ISPs and gateways of large organizations) over the Internet network in an attempt to expand its exposure to as many new files as possible. Widespread deployment will result in a scenario in which almost every new file goes through the framework. If the file is informative enough or is assessed as likely being malicious, it will be acquired for manual analysis. As Fig. 2 shows, the PDF files transported over the Internet are collected and scrutinized within our framework {1}. Then, the "known files module" filters all the known benign and malicious PDF files {2} (according to a white list, reputation systems [27], and the anti-virus signature repository).

Then, only the unknown PDF files from the previous step are checked for their compatibility with PDF specifications (explained in "Dataset" Section that describes the dataset collection) {3}. The incompatible PDF files are immediately blocked (since only compatible files are relevant for organizations and individual users). Note that the compatibility check is extremely important since it blocks many of the malicious files (discussed in "Dataset collection" section) from being introduced to the next step which is more time consuming and, in some cases, might even require manual analysis done by a human expert. The remaining PDF files which are compatible and unknown are then introduced to the next step. In the advanced analysis step {4}, these compatible files are transformed into vector form for the advanced check. In vector form the files are represented as vectors of the frequencies of structural paths of the PDF files (as will be explained in the following sub-section). The detection model within this check scrutinizes the PDF files and provides two values for each file: a classification decision

[21] *Hooking* A technique for intercepting functions calls, messages, or events passed between software components in order to alter an application or operating system behavior.

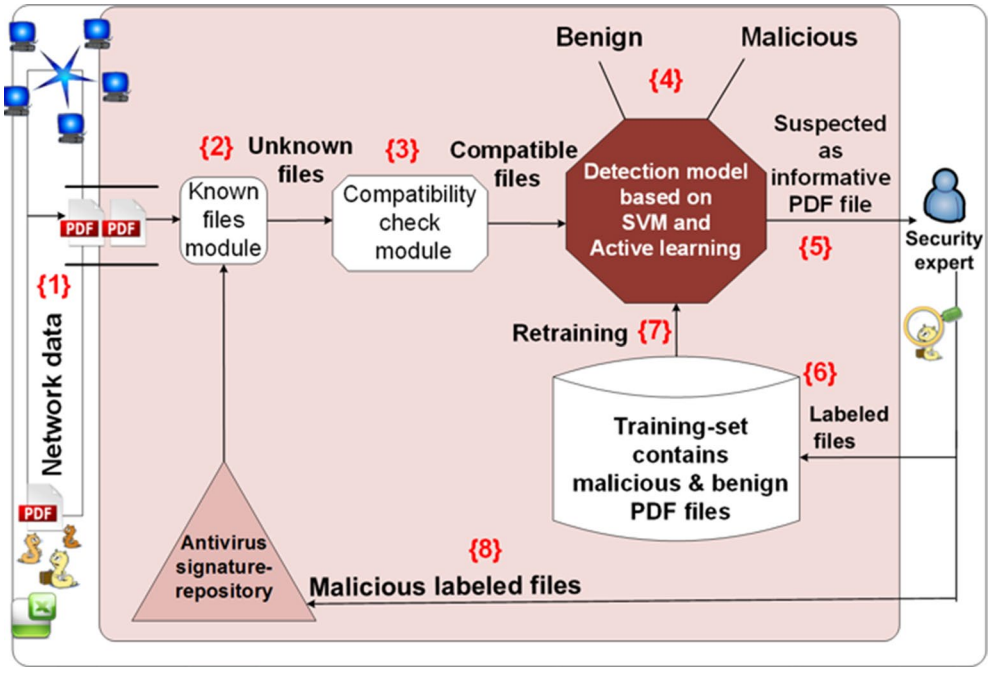

Fig. 2 The process of maintaining the updatability of the anti-virus tool using AL methods

using the SVM (Support Vector Machine) classification algorithm and a distance calculation from the separating hyperplane using Eq. 1. The AL method acquires files which are found to be informative and sends these files to an expert for manual analysis and labeling {5}. These labeled informative files are being added to the training set which is used to induce new and updated detection model. By acquiring these informative PDF files, we aim to frequently update the anti-virus software by focusing the expert's efforts on labeling PDF files that are most likely to be malware or benign PDF files that are expected to improve the detection model. Recall that informative files are those files that when added to the training set improve the detection model's predictive capabilities and enrich the anti-virus signature repository. Accordingly, in our context there are two types of files that may be considered informative. The first type includes files in which the classifier has limited confidence as to their classification (the probability that they are malicious is very close to the probability that they are benign). Acquiring them as labeled examples will probably improve the model's detection capabilities. In practical terms, these PDF files will have new structural paths or special combinations of existing structural paths that represent their execution code (inside the binary code of the executable). Therefore these files will probably lie inside the SVM margin and consequently will be acquired by the SVM-Margin strategy that selects informative files, both malicious and benign, that are a short distance from the separating hyperplane.

The second type of informative files includes those that lie deep inside the malicious side of the SVM margin and are a maximal distance from the separating hyperplane according to Eq. 1. These PDF files will be acquired by the exploitation method (described later) and are also a maximal distance from the labeled files. This distance is measured by the KFF calculation that will be further explained as well. These informative files are then added to the training set {6} for updating and retraining the detection model {7}. The files that were labeled as malicious are also added to the anti-virus signature repository in order to enrich and maintain its updatability {8}. Updating the signature repository also requires an update to clients utilizing the anti-virus application. The framework integrates two main phases: training and detection/updating.

Training

A detection model is trained over an initial training set that includes both malicious and benign PDF files. After the model is tested over a stream that consists only of unknown files that were presented to it on the first day, the initial performance of the detection model is evaluated.

Detection and updating

For every unknown PDF file in the stream, the detection model provides a classification, while the AL method provides a rank representing how informative the file is. The framework will consider acquiring the files based on these two factors. After being selected and receiving their true labels from the expert, the informative PDF files are acquired by the training set, and the signature repository is updated as well (in cases which the files are malicious). The detection model is retrained over the updated and extended training set which now also includes the acquired examples that are regarded as being very informative. At the end of the current day the updated model receives a new stream of unknown files on which the updated model is once again tested and from which the updated model acquires additional informative files. Note that the motive is to acquire as many malicious PDF files as possible, since such information will maximally update the anti-virus tools that protect most organizations.

We employed the SVM classification algorithm using the radial basis function (RBF) kernel in a supervised learning approach. We used the SVM algorithm for the following reasons: (1) SVM has been successfully used to detect worms [25, 28], classify malware into species, and detect zero day attacks [29]; (2) the trained SVM classifier is a black-box that is hard for an attacker to understand [28]; (3) SVM has proven to be very efficient when combined with AL methods [26]; and (4) SVM is known for its ability to handle large numbers of features which makes it suitable for handling the number of structural paths extracted from the PDF files [18]. In our experiments we used Lib-SVM implementation [30] which also supports multiclass classification.

Detection of malicious PDF files using structural paths

In order to detect and acquire unknown malicious PDF files, we implemented a static analysis approach based on the hierarchical structural path feature extraction method presented by Šrndic and Laskov [6]. Instead of analyzing JavaScript code or any other content, this approach makes use of essential differences in the structural properties of malicious and benign PDF files. It parses the PDF files and extracts their structural paths which are the paths in the files' hierarchical object tree that characterize a document's structure. Each structural path is analogous to a set of relations between the objects within the PDF file. For example, the "…/JS" path means that the PDF file contains JavaScript code. The structural paths represent the file's properties and possible actions, acting like the file's genes rather than its current behavior which can be postponed or delayed according to specific conditions. Their detection model was based on these structural paths and was found to be very effective at differentiating between malicious and benign PDF files, even against new unknown malicious files that were created two months after the classification model was built. Šrndic and Laskov reported on high TPR and low FPR. The method was tested against previous detectors: *MDScan* [13], *PJScan* [7], *ShellOS* [20], and *PDFMS* [8], and the comparison clearly demonstrated the efficiency and resilience of this method in the detection of new malicious PDF files over other methods.

Figure 3 provides a simple example of the conversion of a PDF file into a set of structural paths. The PDF code is treated as a tree of objects. Note that only paths of the leaves in the structural tree are counted.

When an attacker injects malicious content into the PDF file, the file structure inevitably changes. Thus, this approach can easily discriminate between benign and malicious files. This approach has several advantages. First, it is not affected by code obfuscation, filtering, and other encryption methods used for hiding and concealing malicious code in the PDF file, since it doesn't actually analyze the embedded JavaScript code. Second, it is

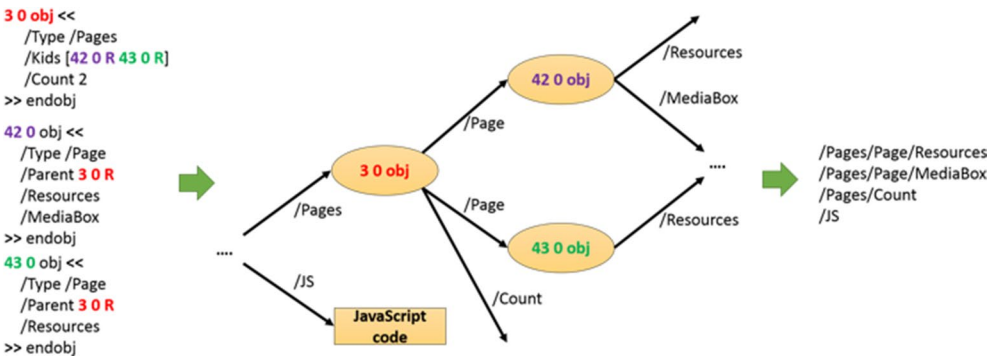

Fig. 3 Example of the conversion of a PDF file to a set of structural paths

robust towards mimicry and reverse mimicry attacks [6]. Finally, it is very fast and lightweight, since the analysis is done statically and does not require execution of the PDF file. Because of this, analysis is conducted quite quickly at the rate of 28 ms for an average file [6].

Selective sampling and active learning methods

Since our framework aims to provide solutions to real problems it must be based on a sophisticated, fast selective sampling method which also efficiently identifies informative files. We compared our proposed AL methods to other selective sampling methods, and all of these methods are described below.

Random selection (random)

While random selection is obviously not an AL method, it is at the "lower bound" of the selection methods discussed. We are unaware of an anti-virus tool that uses an AL method for maintaining and improving its updatability. Consequently, we expect that all AL methods will perform better than a selection process based on the random acquisition of files.

The SVM-Simple-Margin AL method (SVM-Margin)

The SVM-Simple-Margin method [24] (referred to as SVM-Margin) is directly related to the SVM classifier. Using a kernel function, the SVM implicitly projects the training examples into a different (usually a higher dimensional) feature space denoted by F. In this space there is a set of hypotheses that are consistent with the training set, and these hypotheses create a linear separation of the training set. From among the consistent hypotheses, referred to as the version-space (VS), the SVM identifies the best hypothesis with the maximum margin. To achieve a situation where the VS contains the most accurate and consistent hypothesis, the SVM-Margin method selects examples from the pool of unlabeled examples reducing the number of hypotheses. This method is based on simple heuristics that depend on the relationship between the VS and the SVM with the maximum margin, because calculating the VS is complex and impractical where large datasets are concerned. Examples that lie closest to the separating hyperplane (inside the margin) are more likely to be informative and new to the classifier, and these examples are selected for labeling and acquisition.

This method, in contrast to ours, selects examples according to their distance from the separating hyperplane only to explore and acquire the informative files without relation to their classified labels, i.e., not specifically focusing on malware instances. The SVM-Margin method is very fast and can be applied to real problems, yet SVM-Margin's authors [24] indicate that this agility is achieved, because it is based on a rough approximation and relies on assumptions that the VS is fairly symmetric and the hyperplane's Normal (W) is centrally placed, assumptions that have been shown to fail significantly [31]. The method may query instances whose hyperplane does not intersect the VS and therefore may not be informative. The SVM-Margin method for detecting instances of PC malware was used by Moskovitch et al. [32, 33] whose preliminary results found that the method also assisted in updating the detection model but not the anti-virus application itself; however, in this study the method was only used for a one day-long trial. We compare its performance to our proposed AL methods over a longer period, in a daily process of detection and acquisition experiments that better reflect reality. This serves as our baseline AL method, and we expect our method to improve the new malicious PDF detection and acquisition seen in SVM-Margin.

Exploitation: our proposed active learning method

Our method, "Exploitation," is based on SVM classifier principles and is oriented towards selecting examples most likely to be malicious that lie furthest from the separating hyperplane. Thus, our method supports the goal of boosting the signature repository of the anti-virus tool by acquiring as much new malware as possible. For every file X that is suspected of being malicious, Exploitation rates its distance from the separating hyperplane using Eq. 1 based on the Normal of the separating hyperplane of the SVM classifier that serves as the detection model. As explained above, the separating hyperplane of the SVM is represented by W, which is the Normal of the separating hyperplane and is actually a linear combination of the most important examples (supporting vectors), multiplied by LaGrange multipliers (alphas) and by the kernel function K that assists in achieving linear separation in higher dimensions. Accordingly, the distance in Eq. 1 is simply calculated between example X and the Normal (W) presented in Eq. 2.

$$Dist(X) = \left(\sum_{n}^{1} \alpha_i y_i K(x_i x) \right) \tag{1}$$

$$w = \sum_{1}^{n} \alpha_i y_i \Phi(x_i) \tag{2}$$

In Fig. 4 the files that were acquired (marked with a red circle) are those files that are classified as malicious and are at the maximum distance from the separating hyperplane. Acquiring several new malicious files that are very similar and belong to the same virus family is considered a waste of manual analysis resources, since

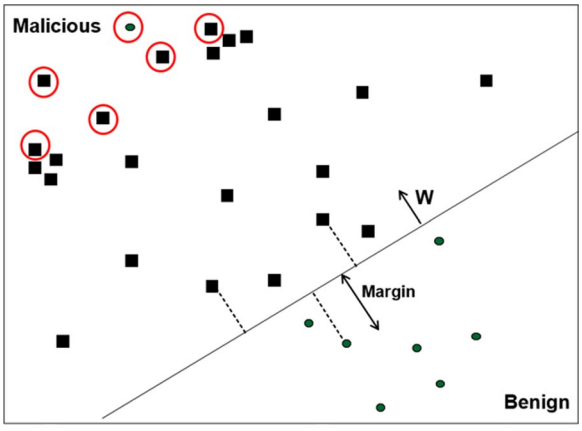

Fig. 4 The criteria by which exploitation acquires new unknown malicious PDF files. These files lie the farthest from the hyperplane and are regarded as representative files

these files will probably be detected by the same signature. Thus, acquiring one representative file for this set of new malicious files will serve the goal of efficiently updating the signature repository. In order to enhance the signature repository as much as possible, we also check the similarity between the selected files using the kernel farthest-first (KFF) method suggested by Baram et al. [34] which enables us to avoid acquiring examples that are quite similar. Consequently, only the representative files that are most likely malicious are selected. In cases in which the representative file is detected as malware as a result of the manual analysis, all its variants that weren't acquired will be detected when the anti-virus is updated. In cases in which these files are not actually variants of the same malware, they will be acquired the following day (after the detection model has been updated), as long as they are still most likely to be malware. In Fig. 4 it can be observed that there are sets of relatively similar files (based on their distance in the kernel space), however, only the representative files that are most likely to be malware are acquired. The SVM classifier defines the class margins using a small set of supporting vectors (i.e., PDF files). While the usual goal is to improve classification by uncovering (labeling) files from the margin area, our primary goal is to acquire malware in order to update the anti-virus. In contrast to SVM-Margin which explores examples that lie inside the SVM margin, Exploitation explores the "malicious side" to discover new and unknown malicious files that are essential for the frequent update of the anti-virus signature repository, a process which occasionally also results in the discovery of benign files (files which will likely become support vectors and update the classifier). Figure 4 presents an example of a file lying far inside the malicious side that was found to be benign. With this method the distance calculation required for each instance is fast and equivalent to the time it takes to classify an instance in a SVM classifier, thus it is applicable for products working in real-time.

Combination: a combined active learning method

The "combination" method lies between SVM-Margin and exploitation. On the one hand, the combination method begins by acquiring examples based on SVM-Margin criteria in order to acquire the most informative PDF files, acquiring both malicious and benign files. This exploration phase is important in order to enable the detection model to discriminate between malicious and benign PDF files. On the other hand, the Combination method then tries to maximally update the signature repository in an exploitation phase, drawing on the exploitation method. This means that in the early acquisition period, during the first part of the day, SVM-Margin leads, compared to exploitation. As the day progresses, Exploitation becomes predominant. However, the Combination method, which consists of first conducting exploration by SVM-Margin and then conducting Exploitation, was also applied in the course of the 10 days of the experiment, and over a period of days, combination will perform more exploitation than SVM-Margin. This means that on the ith day there is more exploitation than on the $(i-1)th$ day. We defined and tracked several configurations over the course of several days. Regarding the relation between SVM-Margin and Exploitation, we found that a balanced division performs better than other divisions (i.e., during the first half of the study, the method will acquire more files using SVM-Margin, while during the second half of the study, exploitation takes the leading role in the acquisition of files). In short, this method tries to take the best from both of the previous methods.

Note that our combination AL method actually repeats in cycles of X days according to the configurations of the administrator, so that in each cycle, the combination method starts with more explorative phase (SVM-Margin) in which both informative malicious and benign PDF files are being acquired. Then, combination conducts more acquisition according to the exploitation AL method in order to acquire more PDF malware and enrich the antivirus signature repository. In our experiments we conducted only one cycle of 10 days experiment, yet on a long term, this strategy will work on a continuous manner in cycles of 10 days, in which every cycle will be both consisted of exploration (SVM-Margin) and exploitation. The administrator have the privilege to decide what will be the length of the cycles.

Table 2 Our collected dataset categorized as malicious, benign, and incompatible PDF files (bracketed)

Dataset source	Year	Malicious files	Benign files
VirusTotal[a] repository	2012–2014	17,596 (1017)	–
Srndic and Laskov [6]	2012	27,757 (437)	–
Contagio project	2010	410 (175)	–
Internet and Ben-Gurion University (random selection)	2013–2014	0	5145
Total		*45,763 (1629)*	*5145*

[a] https://www.virustotal.com/

Evaluation

In this section we present the dataset on which we based our experiments and elaborate on the composition of the files within the dataset. We also discuss our insights regarding the compatibility of the PDF files. At the conclusion of this section we present our experimental design which was aimed at comprehensively evaluating our framework.

Dataset

Dataset collection

We created a collection of benign and malicious PDF files. We acquired a total of 50,908 PDF files consisting of 45,763 malicious and 5145 benign files. The malicious PDF files aimed at compromising the Windows operating system, the most commonly attacked system used by organizations. The files were collected during the years 2012–2014 from the four sources presented in Table 2. All the files were verified to be labeled correctly as malicious or benign using Kaspersky[22] anti-virus.

The benign files were collected from Ben-Gurion University, a very diverse organization that generates an enormous number of PDF files from diverse sources, including many different academic and administrative departments. Although the entire benign collection originated from one institution, it should be noted that it is actually quite varied and contains PDF files that are substantially different from one another in their content, elements, and purpose that were randomly selected from the following sources within the university: academic papers, automatically generated reports, student assignments, previous exams, administrative forms (also with active functionality), architectural renderings and planning documentation, brochures, and newsletters.

In our recent studies [4], [35] we thoroughly investigated PDF files and found that most malicious PDF files (96.5 %) are not compatible with the PDF file format specifications (checked with the Adobe PDF Reference[23]). These files cannot be viewed by the PDF reader or the user, and an error message will likely be displayed on the screen when an attempt is made to open the file. However in such cases involving incompatible malicious PDF files, the malicious operations will still be executed. A common incompatibility observed was located at the end of the file between the "startxref" and " %EOF" lines. This line should contain a number serving as a reference (offset) to where the last cross reference table section is located in the file. In cases of incompatibility, the number that appears is incorrect.

We would like to reiterate that incompatible files are unreadable, because once they are opened they either cause the PDF reader to crash or generate an error message sent to the user regarding the corruption of the file. However, in cases in which the file is malicious, the malicious operation is likely to occur whether or not the file has been presented the PDF reader. Regardless as to whether the incompatible file is benign or malicious, the file cannot be viewed by the user. Thus, there is no reason to transfer such files to the user, and we suggest that they get blocked. We observed that 96.5 % of the malicious files within our large and representative data collection of more than 45,000 malicious PDF files were incompatible; based on this, we can therefore assume that 96.5 % of all malicious PDF files are incompatible. Because of this, when one encounters an incompatible PDF file, there is a strong chance that it is, in fact, a malicious PDF file.

Our proposed framework uses these observations to flag files that can initially be blocked from the network of an organization or private computer, since they cannot be opened correctly; therefore our ML based framework was applied only to the compatible files (6774). Table 2 presents the number of compatible files in each of our collections (bracketed).

Figure 5 presents a general breakdown of the threats raised by the compatible malicious PDF files within our dataset.

As can be seen, 3 % of the 1629 malicious files were classified as containing a Trojan which is a malicious program that when executed performs covert actions that have not been permitted by the user. 97 % of the malicious files were identified as exploiting some vulnerability in one or more PDF readers. More specifically, they

[22] http://www.kaspersky.com.

[23] http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf.

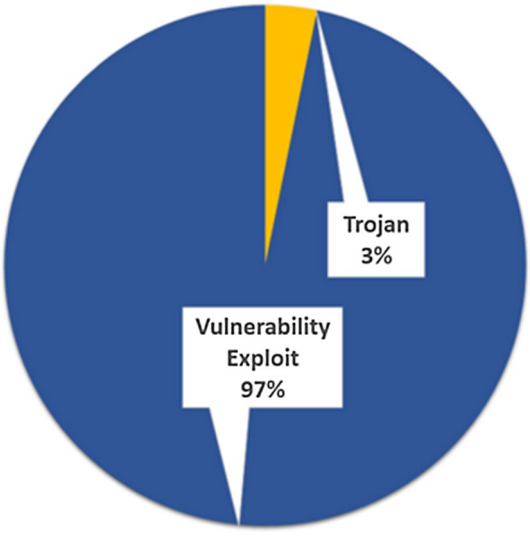

Fig. 5 Breakdown of the threats identified among the malicious PDF files found in our dataset

exploited 23 unique vulnerabilities, including the following common vulnerability and exposures (CVE[24]): CVE-*2007*-3845, CVE-*2008*-2551, CVE-*2009*-0927, CVE-*2010*-0188, and CVE-*2013*-0640. The four digits after the "CVE-"represent the year the vulnerability was discovered. As can be seen, our dataset contains malicious files that exploit cross-time vulnerabilities. It is interesting to note that despite the fact that our dataset consists of malicious PDF files that were created from 2012 to 2014, the files also exploited much older vulnerabilities, including some discovered during 2007–2011. This indicates that attackers are aware of the fact that many organizations are not up-to-date with the newest versions of software and thus are exposed to older vulnerabilities already discovered, as well as new unknown vulnerabilities. In Table 3 we provide brief details regarding the most commonly exploited vulnerabilities mentioned above. The table indicates the percentage each vulnerability represents of the total dataset.

The percent of malicious PDF files in the final dataset after removing incompatible files is 24 %. This percentage is likely higher than the actual percentage of malicious PDF files found on the web. According to our recent study [36] that dealt with the issue of imbalanced datasets in the cyber-security domain, the best detection performance is achieved when the percentage of malicious PDF files in the training set is equal to the percentage in the test set. We have adhered to this guideline, thus the percentage should not affect the detection and updatability capabilities reported upon now. We were unable to find any reliable source that published an accurate percentage of malicious PDF files on the web. Therefore we will rely upon the results of our recent study [36], and when the exact percentage is determined, we will adjust both the training and test sets to that percentage. In addition, in Table 4 we also included the percentage of malicious PDF files in previous and relevant studies in the malicious PDF detection domain. We have done so in order to compare the way this was handled in previous studies and to demonstrate that none of the previous studies addressed this issue or seemed to adhere to a figure representing the actual percentage of malicious PDF files on the web. As can be seen, the average percentage used within these studies was 38 % which is very high and not likely reasonable. It is important to understand that many of the malicious files on the web haven't been discovered yet, since anti-virus tools are very limited in their detection capabilities and discover only relatively similar variants of known malwares; in addition, it takes time for anti-virus vendors to discover new 0-day malicious files. Therefore our percentage (24 %) likely represents the upper reasonable boundary of the actual percentage of malicious PDF files. Moreover, in our previous study [26], aimed at the detection of malicious executables using ML and AL methods, we estimated the malicious executables percentage to be approximately 9 %. In that study we therefore adjusted our dataset to 9 % malicious files, and our methods worked very well, providing high true positive rates (TPR) and low false positive rate (FPR) rates. Thus, we assume that these methods will be effective in the current study as well.

Dataset creation

As was explained previously, in order to detect and acquire unknown malicious PDF files, we implemented a static analysis approach based on the hierarchical structural path feature extraction method presented by Šrndic and Laskov [6]. Instead of analyzing JavaScript code or any other content, this approach makes use of essential differences in the structural properties of malicious and benign PDF files. Using the PdfFileAnalyzer[25] parser we parsed the compatible PDF files (both malicious and benign, totaling 6774) and extracted all 7963 unique paths that were found within our dataset. Each of these paths was used as feature. We didn't apply a feature selection method, since this number of features can easily be handled by the SVM classifier used in our experiments. Each PDF file was represented as a vector of Boolean features so that the presence (1) or absence (0) of a

[24] https://cve.mitre.org/.

[25] http://www.codeproject.com/Articles/450254/PDF-File-Analyzer-With-Csharp-Parsing-Classes-Vers.

Table 3 List of the most exploited vulnerabilities in our dataset

The CVE	Description	Percentage
CVE-*2007*-3845[a]	The version of some PDF readers was found to allow remote attackers to execute arbitrary commands via certain vectors associated with launching malicious code based on the file extension at the end of the URI	49
CVE-*2008*-2551[b]	The DownloaderActiveX Control in Icona SpA C6 Messenger allows remote attackers to force the download and execution of arbitrary files via a URL in the prop download url parameter with the propPost download action parameter set to "run"	4
CVE-*2009*-0927[c]	Stack-based buffer overflow in some adobe reader versions allows remote attackers to execute malicious code via a crafted argument to the *'getIcon'* method of a *'Collab'* object. This executed code can exfiltrate sensitive data to a remote server where it can download and execute dangerous payload to the host	5
CVE-*2010*-0188[d]	Unspecified vulnerability in adobe reader and acrobat allows attackers to cause a denial of service (application crash) or possibly execute malicious code via unknown vectors	6
CVE-*2013*-0640[e]	Adobe reader and acrobat versions allow remote attackers to execute malicious code or cause a denial of service (memory corruption)	32

[a] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-3845

[b] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2551

[c] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-0927

[d] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0188

[e] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-0640

Table 4 Percentage of malicious PDF files in previous studies

Study	Year	Core of the study	Malicious files percentage
[13]	2011	Dynamic analysis of embedded JavaScript code	9
[7]	2011	Static lexical analysis of embedded JavaScript code	94
[20]	2011	Dynamic analysis of embedded JavaScript code	69
[9]	2012	Static tokenization of embedded JavaScript code	43
[19]	2012	Dynamic analysis of embedded JavaScript code	65
[16]	2012	Static analysis of meta-features	5
[8]	2012	Static analysis of embedded keywords	53
[17]	2013	Static analysis using entropy and n-gram	58
[21]	2013	Dynamic analysis of embedded JavaScript code	29
[6]	2013	Static analysis of structural features	12
			Average 38

structural path within a PDF file is represented by 1 or 0 respectively.

Experimental design

The objective in our main experiment was to evaluate and compare the performance of our new AL methods to the existing selection method, SVM-Margin, on two tasks:

- Acquiring as many new unknown malicious PDF files as possible on a daily basis in order to efficiently enrich the signature repository of the anti-virus
- Updating the predictive capabilities of the detection model that serves as the knowledge store of AL methods and improving its ability to efficiently identify the most informative new malicious PDF files

Over a 10-day period, we compared PDF file acquisition based on AL methods to random selection based on the performance of the detection model. In our acquisition experiments we used 6774 compatible PDF files (5145 benign and 1629 malicious) in our repository and created 10 randomly selected datasets as each one of them contains 10 subsets of 620 files representing each day's stream of new files. The 574 remaining files were used by the initial training set to induce the initial model. Note that each day's stream contained 620 PDF files. At first we induced the initial model by training it over the 574 known PDF files. We then tested it on the first day's stream. Next, from this same stream, the selective sampling method selected the most informative PDF files according to that method's criteria. The informative files were sent to an expert who labeled them. The files were later acquired by the training

set which was enriched with an additional X new informative files. When a file was found to be malicious, it was immediately used to update the signature repository of the anti-virus, and an update was also distributed to clients. The process was repeated over the next 9 days. The performance of the detection model was averaged for 10 runs over the 10 different datasets that were created. Each selective sampling method was checked separately on 20 different acts of file acquisition (each consisting of a different number of PDF files). This means that for each act of acquisition, the methods were restricted to acquiring a predefined number of files equal to the amounts that followed, denoted as X: 10 files, 20 files, and so on (with gaps of 10 files), until 160 files, and subsequently at 200, 250, 300 and 350 files. We also considered the acquisition of all the files in the daily stream (referred to as the ALL method), which represents an ideal, but impractical way, of acquiring all the new files, and more specifically, all the malicious PDF files.

The experiment's steps are summarized as follows

1. Inducing the initial detection model from the initial available training set, i.e., training set available up to day d (the initial training set includes 574 PDF files)
2. Evaluating the detection model on the stream of day $(d+1)$ to measure its initial performance
3. Introduction of the stream of day $(d+1)$ to the selective sampling method which chooses the X most informative files according to its criteria and sends them to the expert for manual analysis and labeling
4. Acquiring the informative files and adding them to the training set, as well as using their extracted signature to update the anti-virus signature repository
5. Inducing an updated detection model from the updated training set and applying the updated model on the stream of the next day $(d+2)$

This process repeats itself from the first day until the tenth day.

Results

We rigorously evaluated the efficiency and effectiveness of our framework, comparing four selective sampling methods: (1) a well-known existing AL method, termed SVM-Simple-Margin (SVM-Margin) and based on [24]; our proposed methods (2) exploitation and (3) combination; and (4) random-selection (random) as a "lower bound". Each method was checked for all 20 acquisition amounts and the results were the mean of 10 different folds. Due to space limitations we have depicted the results of the most representative acquisition amount of 160 PDF files which is equal to the maximal mean number of PDF files found in the daily stream.

We now present the results of the core measure in this study, the number of new unknown malicious files that were discovered and finally acquired into the training set and signature repository of the anti-virus software. As explained above, each day the framework deals with 620 new PDF files consisting of about 160 new unknown malicious PDF files. Statistically, the more files that are selected daily, the more malicious files will be acquired daily. Yet, using AL methods, we tried to improve the number of malicious files acquired by means of existing solutions. More specifically, using our methods (exploitation and combination) we also sought to improve the number of files acquired by SVM-Margin.

Figure 6 presents the number of malicious PDF files obtained by acquiring 160 files daily by each of the four methods during the course of the 10-day experiment. Exploitation and combination outperformed the other selection methods. Exploitation was the only one that showed an increasing trend from the first day, while combination showed a decrease in the second day and then exhibited an increasing trend on the following days. Thus, from days four to 10, it performed like exploitation, and during these days they both outperformed the other methods (SVM-Margin and random selection). Exploitation was successful at acquiring the maximal number of malwares from the 160 files acquired daily and to support the results presented in Fig. 6 and our claim, we performed a single-factor Anova statistical test on the acquisition rates achieved by the exploitation and combination methods. The Anova tests between the AL methods provided P values lower than the 5 % (alpha) of the significance level; thus the difference between the exploitation and combination are statistically significant, confirming that exploitation AL method performed better than combination in terms of number of malicious PDF files acquired.

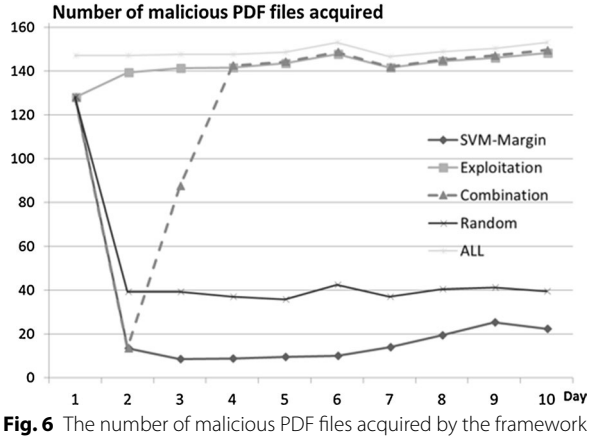

Fig. 6 The number of malicious PDF files acquired by the framework for different methods with acquisition of 160 files daily

On the first day, the number of new malicious PDF files is 128 since the initial detection model was trained on an initial set of 574 labeled PDF files that contained 128 malwares. We decided to induce the initial detection model on 574 files in order to have a stable detection model with sufficient detection performance from the start (92.5 % TPR on the first day) that can still be improved through our AL based framework.

The superiority of exploitation over combination can be observed on the second and third days. During these 2 days, combination conducts more exploration in regarding to acquisition of informative PDF files (both malicious and benign), therefore acquired much less malicious PDF files than the exploitation method that is oriented towards the malicious PDF files acquisition. On the second day, exploitation acquired 10 times more PDF malware than combination (140 compare to 14 malicious PDF files), and on the third day exploitation acquired 1.6 times more malicious PDF files than combination (141 compare to 87). Then, from the fourth day, combination performed as well as exploitation in regarding to malicious PDF files acquisition, since the exploitation phase of combination have become more dominant along the days."

On the tenth day, using combination and exploitation, 93.75 and 92.5 % of the acquired files were malicious (150 and 148 files out of 160); using SVM-Margin, only 13.5 % of the acquired files were malicious (22 files out of 160 which is even less than random). This presents a significant improvement of almost 80 % in unknown malware acquisition. Note that on the tenth day, using Random, only 25 % of the acquired PDF files were malicious (40 files out of 160). This is far less than the malware acquisition rates achieved by both combination and exploitation. The trend is very clear from the second day at which point combination and exploitation each acquired more malicious files than the day before—a finding that demonstrates the impact of updating the detection model, identifying new malwares, and enriching the signature repository of the anti-virus. Moreover, because they are different, the acquired malwares are also expected to be of higher quality in terms of their contribution to the detection model and signature repository.

As far as we could observe, the random selection trend was constant, with no improvement in acquisition capabilities over the 10 days. While the SVM-Margin method showed a decrease in the number of malwares acquired through the fifth day, it showed a very negligible improvement from the sixth day. It can be seen that the performance of our methods was much closer to the ALL line which represents the maximum number of malicious PDF files that can be acquired each day. This phenomenon can be explained by looking at the ways in which the methods work. The SVM-Margin acquires examples about which the detection model is less confident. Consequently, they are considered to be more informative but not necessarily malicious. As was explained previously, SVM-Margin selects new informative PDF files inside the margin of the SVM. Over time and with improvement of the detection model towards more malicious files, it seems that the malicious files are less informative (due to the fact that malware writers frequently try to use upgraded variants of previous malwares). Since these new malwares might not lie inside the margin, SVM-Margin may actually be acquiring informative benign, rather than malicious, files. However, our methods, combination and exploitation, are more oriented toward acquiring the most informative files and most likely malware by obtaining informative PDF files from the malicious side of the SVM margin. As a result, an increasing number of new malwares are acquired; in addition, if an acquired benign file lies deep within the malicious side, it is still informative and can be used for learning purposes and to improve the next day's detection capabilities.

Our AL methods outperformed the SVM-Margin method and improved the capabilities of acquiring new PDF malwares and enriching the signature repository of the anti-virus software. In addition, compared to SVM-Margin, our methods also maintained the predictive performance of the detection model that serves as the knowledge store of the acquisition process.

Figure 7 presents the TPR levels and their trends during the course of the 10-day study. SVM-Margin outperformed other selection methods in the TPR measure, while our AL methods, combination and exploitation, came close to SVM-Margin (SVM-Margin achieved 1 % better TPR rates than combination and 2 % better than exploitation) and performed better than random. To support the results presented in Fig. 7 and our claim, we performed several single-factor Anova statistical tests on the TPR rates achieved by the three AL methods and random. The Anova test between the three AL methods provided P-Values higher than the 5 % (alpha) of the significance level; thus the difference between the AL methods is not statistically significant, confirming that our AL methods performed as well as the exploration method in terms of predictive capabilities. While the Anova tests between the AL methods and random provided P values lower than the 5 % (alpha) of the significance level; thus the difference between the AL methods and random is statistically significant, confirming that our AL methods performed better than random also in terms of predictive capabilities. In addition, the performance of the detection model improves as more files are acquired daily, so that on the tenth day of the experiment, the results indicate that by only acquiring a small and well selected set

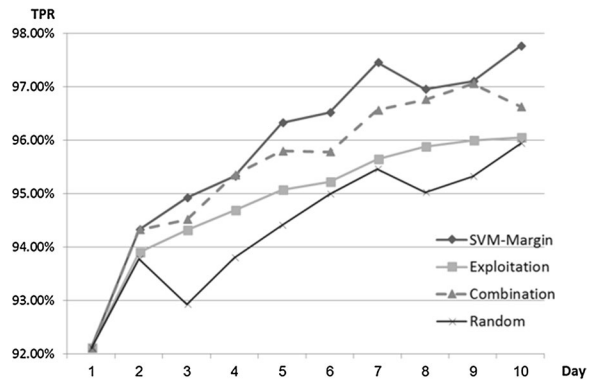

Fig. 7 The TPR of the framework over the period of 10 days for different methods through the acquisition of 160 PDF files daily

of informative files (25 % of the stream), the detection model can achieve TPR levels (97.7 % with SVM-Margin, 96.7 % with combination, and 96.05 % with exploitation) that are quite close to those achieved by acquiring the whole stream (98.4 %).

Figure 8 presents the FPR levels of the four acquisition methods. As can be observed, the FPR rates were low and quite similar among the AL methods. A similar decreasing FPR trend began to emerge on the second day. This decrease indicates an improvement in the detection capabilities and the contribution of the AL methods, in contrast to the increase in FPR rates observed in random from the fifth day. Random had the highest FPR in the course of the 10-day period, which indicates its inability to select more informative files over the 10 days.

Apart from the second day, combination and exploitation achieved quite similar FPR rates which were slightly higher than SVM-Margin. To support the results presented in Fig. 8 and our claim, we performed several single-factor Anova statistical tests on the FPR rates

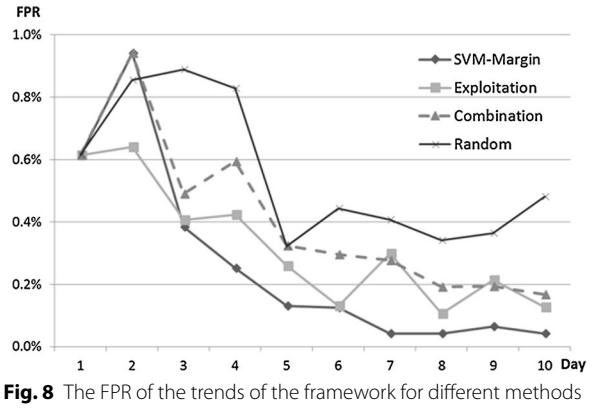

Fig. 8 The FPR of the trends of the framework for different methods based on acquiring 160 PDF files daily

achieved by the three AL methods and Random. The Anova test between the three AL methods provided P values higher than the 5 % (α) of the significance level; thus the difference between the AL methods is not statistically significant, confirming that our AL methods performed as well as the exploration method in terms of false positives of the predictive capabilities. While The Anova tests between the AL methods and random provided P values lower than the 5 % (α) of the significance level; thus the difference between the AL methods and random is statistically significant, confirming that our AL methods performed better than random also in terms of false positives of the predictive capabilities.

On the tenth day, combination and exploitation had an FPR of 0.1 %, while SVM-Margin had an FPR of 0.05 %. This indicates that our methods, exploitation and combination, performed as well as the SVM-Margin method with regard to predictive capabilities (TPR and FPR) but much better than SVM-Margin in acquiring a large number of new PDF malwares daily and in enriching the signature repository of the anti-virus. On each day of the acquisition iteration, we evaluated the learned classifier, and the FPR is presented for the 10-day period. A set of new unknown files, both malicious and benign, is presented to the classifier each day, thus the FPR does not constantly decrease which would occur if the classifier was tested on the same files each day.

We now compare the detection rate of our framework with the leading anti-virus tools (those with the top TPR rates) commonly used by organizations, utilizing the malicious PDF files within our experimental dataset. We used VirusTotal[26] an anti-virus service that include many different anti-virus tools in order to evaluate the detection rates.

Figure 9 shows that the most accurate anti-virus, AntiVir, had a detection rate of 92.5 %, while our methods outperformed all of the anti-viruses in the task of detecting new malicious PDF files. Using our full framework, including the SVM based detection model, the AL methods, and the enhancement process, we achieved a TPR of 97.7 % using only 25 % of labeled data daily (160 PDF files out of 620), which means a reduction of 75 % in security experts' efforts. In a nutshell, our AL based framework was able to better induce an updated detection model on a daily basis, outperforming all the anti-virus tools and managing to accomplish this using only a fraction of the new PDF files (25 %)—the most informative portion, consisting of the most valuable information required for updating the knowledge stores of the detection model and anti-virus tools.

[26] http://www.virustotal.com.

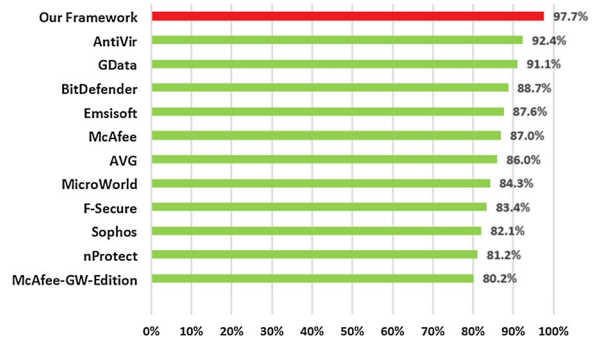

Fig. 9 The TPR of the framework compared to anti-viruses commonly used by organizations

These results demonstrate the efficiency of the framework in maintaining and improving the updatability of the detection model and ultimately of the anti-virus tool. These factors also demonstrate the benefits obtained by performing this process on a daily basis—benefits which will likely include economic rewards as well.

Discussion and conclusion

We presented a framework for efficiently updating anti-virus tools with unknown malicious PDF malware files. With our updated classifier, we can better detect new malicious PDF files that can be utilized to maintain an anti-virus tool. Due to the mass creation of new files, especially new malicious PDF files, both the anti-virus and the detection model (classifier) must be updated with new and labeled PDF files. Such labeling is done manually by human experts, thus the goal of the AL component is to focus expert efforts on labeling PDF files that are more likely to be malicious or on PDF files that might add new information about benign files. Our proposed framework is based on our AL methods (exploitation and combination), specially designed for acquiring unknown malware. The framework seeks to acquire the most informative PDF files, benign and malicious, in order to improve classifier performance, enabling it to frequently discover and enrich the signature repository of anti-virus tools with new unknown malware.

In general, three of the AL methods performed very well at updating the detection model, with our methods, combination and exploitation, outperforming SVM-Margin in the main goal of the study which is the acquisition of new unknown malicious PDF files. The evaluation of the classifier before and after the daily acquisition showed an improvement in the detection rate, and subsequently more new malwares were acquired. On the tenth day, Combination acquired almost seven times more PDF malwares (150) than the number acquired by SVM-Margin (22 PDF malwares) and almost four times more PDF malwares than those acquired by the Random method (40 PDF malwares). While our combination and exploitation methods showed an increasing trend in the number of PDF malwares acquired in the course of the 10 days, SVM-Margin showed unstable and poor performance, and random was consistent. Therefore our framework was found to be effective at updating the anti-virus software by acquiring the maximum number of malicious PDF files. It also maintained a well updated model that is aimed at daily detection of new and unknown malicious PDF files.

One of our authors is a security expert who works as a virus analyst at one of the known anti-virus companies. According to his experience it requires up to 30 min for a virus analyst to determine whether a file is malicious or benign using both static and dynamic tools. Therefore, our approach was aimed at focusing the experts' efforts on the most informative files. The manual labeling efforts equals the number of files acquired daily multiplied by the analysis time (30 min per file). This means that if 160 files were acquired per day, 80 h would be required each day to analyze all the files. This is the equivalent of ten security experts, a reasonable number of analysts for an anti-virus company. Anti-virus companies can use this framework, adjust it accordingly to suit their needs and resources, and thus acquire the desired number of files for analysis. It is also worth noting some of the advantages of our AL methods compared to SVM-Margin and passive learning. First, using our AL methods it is possible to acquire fewer files while achieving nearly the same detection capabilities as other methods; other methods must acquire a larger number of malicious files to achieve poorer results. Second, our AL methods acquired a greater number of malicious PDF files daily than the alternative solutions; in so doing, updating the anti-virus software became more efficient.

Our developing framework is currently based on a feature extractor tailored to PDF files (structural paths as previously discussed), and consequently our framework is limited to providing updating solutions and detection capabilities for attacks that affect the structural paths within PDF files. Implementing and integrating more feature extractors within the framework will result in more robust detection and updatability capabilities. An additional limitation is the fact that the framework can only provide solutions for PDF files, however there are many other widely-used document types such as Microsoft office files (e.g., *.docx, *.xlsx, *.pptx, *.rtf) that have become popular means for launching cyber-attacks aimed at compromising organizations. These types of

files are substantially different from PDF files, and thus the framework must be adapted to cope with the challenges they pose.

In future work, in addition to additional types of malicious documents we are interested in extending this framework to Android applications. Due to their resource limitations, mobile devices are extremely dependent on anti-virus solutions that are frequently and efficiently updated. Quite possibly our suggested AL framework could address this problem.

Authors' contributions

NN initiated and led the study and was the main author of the manuscript. His parts included determining the problem, designing the solution and emperical experiments, analyzing the results and providing the insights. He especially focused on developing the AL framework and AL methods applied in the study. AC focused on the related work done in the malicious PDF detection domain, its background and provided comprehensive explanations regarding to the PDF file's structure and the attacks that can be carried out through it, AC also participated in the collection of the PDF files (malicious and benign) and revising the manuscript. RM participated in shaping the AL framework, as well as revising the manuscript. AS participated in the design of the experiments and in the collection of the malicious PDF files, as well as revising the manuscript. ME participated in the collection of the malicious PDF files and in extracting the structural features from the PDF files and as well as helped to draft the manuscript. OB participated in the collection of the malicious PDF files, provided insights regarding to efficinet etractio of the features as well as participated in revising the manuscript. YE participated in determining the problem, designing the solution and experiments as well as providing insights and significant information regarding to the need of such a framework in AV companies. All authors read and approved the final manuscript.

Author details

[1] Department of Information Systems Engineering, Ben-Gurion University of the Negev, Beersheba, Israel. [2] Department of Biomedical Informatics, Columbia University, New York, USA.

Acknowledgements

This research was partly supported by the National Cyber Bureau of the Israeli Ministry of Science, Technology and Space. Thanks to Šrndic and Laskov from the University of Tübingen for sharing their malicious PDF dataset with us. We would like to thank *VirusTotal* for granting us access to their private services.

Competing interests

The authors declare that they have no competing interests.

Received: 5 March 2015 Accepted: 19 January 2016
Published online: 18 February 2016

References

1. Christodorescu M, Jha S (2004) Testing malware detectors. ACM SIGSOFT Softw Eng Notes 29(4):34–44
2. White SR, Swimmer M, Pring EJ, Arnold WC, Chess DM, Morar JF (1999) Anatomy of a commercial-grade immune system. IBM Research White Paper
3. Kiem H, Thuy NT, Quang, TMNA machine learning approach to anti-virus system (2004) Joint Workshop of Vietnamese Society of AI, SIGKBS-JSAI, ICS-IPSJ and IEICE-SIGAI on Active Mining 61–65, 4–7 December, Hanoi-Vietnam
4. Nissim N, Cohen A, Glezer C, Elovici Y (2015) Detection of malicious PDF files and directions for enhancements: a state-of-the art survey. Comput Secur 48:246–266
5. Maiorca D, Corona I, Giacinto G (2013) Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious PDF files detection. Presented at Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security
6. Šrndic N, Laskov P (2013) Detection of malicious pdf files based on hierarchical document structure. 20th Annual Network and Distributed System Security Symposium
7. Laskov P, Šrndić N (2011) Static detection of malicious JavaScript-bearing PDF documents. 27th Annual Computer Security Applications Conference
8. Maiorca D, Giacinto G, Corona I (2012) "A pattern recognition system for malicious pdf files detection," in Machine Learning and Data Mining in Pattern Recognition Anonymous
9. Vatamanu C, Gavriluţ D, Benchea R (2012) A practical approach on clustering malicious PDF documents. J Comput Virol 8(4):151–163
10. Baccas P (2010) Finding rules for heuristic detection of malicious pdfs: With analysis of embedded exploit code. 12–12
11. Kittilsen J Detecting malicious PDF documents
12. Hamon Valentin (2013) Malicious URI resolving in PDF documents. J Comput Virol Hacking Tech 9(2):65–76
13. Tzermias Z, Sykiotakis G, Polychronakis M, Markatos EP (2011) Combining static and dynamic analysis for the detection of malicious documents. 4th European Workshop on System Security
14. Liu D, Wang H, Stavrou A (2014) Detecting malicious javascript in pdf through document instrumentation. In Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on 100–111 IEEE
15. Igino Corona, Davide Maiorca, Davide Ariu, Giorgio Giacinto (2014) Lux0R: Detection of Malicious PDF-embedded JavaScript code through Discriminant Analysis of API References. InProceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop (AISec '14). ACM, New York, NY, USA, 47–57. doi: 10.1145/2666652.2666657
16. Smutz C, Stavrou A (2012) Malicious PDF detection using metadata and structural features. 28th Annual Computer Security Applications Conference
17. Pareek H et al (2013) Entropy and n-gram analysis of malicious PDF documents. Int J Eng Res Tech 2(2)
18. Joachims T (1999). Making large scale SVM learning practical
19. Schmitt F, Gassen J, Gerhards-Padilla E (2012) PDF scrutinizer: Detecting JavaScript-based attacks in PDF documents. Tenth Annual International Conference on Privacy, Security and Trust (PST)
20. Snow KZ, Krishnan S, Monrose F, Provos N (2011) SHELLOS: Enabling fast detection and forensic analysis of code injection attacks. USENIX Security Symposium
21. Lu X, Zhuge J, Wang R, Cao Y, Chen Y (2013) De-obfuscation and detection of malicious PDF files with high accuracy. 46th Hawaii International Conference on System Sciences (HICSS)
22. Marco Cova, Christopher Kruegel, Giovanni Vigna (2010) Detection and analysis of drive-by-download attacks and malicious JavaScript code. In Proceedings of the 19th international conference on World wide web (WWW '10). ACM, New York, NY, USA, 281–290. doi: 10.1145/1772690.1772720
23. Maass M, Scherlis WL, Aldrich J (2014) In-nimbo sandboxing. InProceedings of the 2014 Symposium and Bootcamp on the Science of Security (p. 1). ACM
24. Tong S, Koller D (2000–01) Support vector machine active learning with applications to text classification. J Mach Learn Res 2:45–66
25. Nissim N, Moskovitch R, Rokach L, Elovici Y (2012) Detecting unknown computer worm activity via support vector machines and active learning. Pattern Analysis and Applications 15(4):459–475
26. Nissim N, Moskovitch R, Rokach L, Elovici Y Novel Active Learning Methods for Enhanced PC Malware Detection in Windows OS, Expert Systems with Applications, http://dx.doi.org/10.1016/j.eswa.2014.02.053
27. Jnanamurthy HK, Chirag Warty, Sanjay Singh (2013) "Threat Analysis and Malicious User Detection in Reputation Systems using Mean Bisector Analysis and Cosine Similarity (MBACS)"
28. Wang X, Yu W, Champion A, Fu X, Xuan D (2007) "Worms via mining dynamic program execution," Third International Conference Security and Privacy in Communication Networks and the Workshops, SecureComm 412–421
29. Chen Z, Roussopoulos M, Liang Z, Zhang Y, Chen Z, Delis A (2012) Malware characteristics and threats on the internet ecosystem. J Syst Softw 85(7):1650–1672

30. Chang CC, Lin CJ (2011) LIBSVM: a library for support vector machines. TIST 2(3):27
31. Herbrich Ralf (2001) Thore Graepel, and Colin Campbell. "Bayes point machines". J Mach Learn Res 1:245–279
32. Moskovitch R, Nissim N, Elovici Y (2009) Malicious code detection using active learning. In Privacy, Security, and Trust in KDD Springer Berlin Heidelberg; 74–91
33. Moskovitch R, Nissim N, Elovici Y (2007) "Malicious Code Detection and Acquisition Using Active Learning". IEEE International Conference on Intelligence and Security Informatics (IEEE ISI-2007), Rutgers University, New Jersey, USA
34. Baram Y, El-Yaniv R, Luz K (2004) Online choice of active learning algorithms. J Mach Learn Res 5:255–291
35. Nir Nissim, Aviad Cohen, Robert Moskovitch, Oren Barad, Mattan Edry, Assaf Shabtai, Yuval Elovici (2014) "ALPD: Active Learning framework for Enhancing the Detection of Malicious PDF Files aimed at Organizations" JISIC
36. Moskovitch R, Stopel D, Feher C, Nissim N, Japkowicz N, Elovici Y (2009) Unknown malcode detection and the imbalance problem. J Comput Virol 5(4):295–308