Learn2Evade: Learning-Based Generative Model for Evading PDF Malware Classifiers

Ho Bae, Younghan Lee, Yohan Kim, Uiwon Hwang, Sungroh Yoon, *Senior Member, IEEE*, and Yunheung Paek, *Member, IEEE*

***Abstract*—Recent research has shown that a small perturbation to an input may forcibly change the prediction of a machine learning (ML) model. Such variants are commonly referred to as *adversarial examples*. Early studies have focused mostly on ML models for image processing and expanded to other applications, including those for malware classification. In this article, we focus on the problem of finding adversarial examples against ML-based portable document format (PDF) malware classifiers. We deem that our problem is more challenging than those against ML models for image processing because of the highly complex data structure of PDF and of an additional constraint that the generated PDF should exhibit malicious behavior. To resolve our problem, we propose a variant of *generative adversarial networks* that generate evasive variant PDF malware (without any crash), which can be classified as benign by various existing classifiers yet maintaining the original malicious behavior. Our model exploits the target classifier as the second discriminator to rapidly generate an evasive variant PDF with our new feature selection process that includes unique features extracted from malicious PDF files. We evaluate our technique against three representative PDF malware classifiers (Hidost'13, Hidost'16, and PDFrate-v2) and further examine its effectiveness with AntiVirus engines from VirusTotal. To the best of our knowledge, our work is the first to analyze the performance against the commercial AntiVirus engines. Our model finds, with great speed, evasive variants for all selected seeds against state-of-the-art PDF malware classifiers and raises a serious security concern in the presence of adversaries.**

***Impact statement*—PDF has been one of the most popular media to conceal adversarial contents for many years. The reason being that adversaries can exploit the complex structure of PDF in their favor by hiding malicious content. In 2019, more than 73k PDF-based attacks were reported in one month, which accounts for 17% of newly detected threats. With increasing popularity, many ML-based techniques have been proposed for PDF malware classifiers. Such defense mechanisms include support vector machine and random forest classification models trained with a structural map of PDF (Hidost'13 and Hidost'16). Furthermore, ensemble training has been applied with metadata collected from PDF as the training data (PDFrate-v2). In recent studies, many researchers have attempted and succeeded in generating evasive PDF malware (adversarial examples) that bypass such defense techniques. However, the current method heavily relies on a random mutation algorithm resulting in repeated computation for a significant period of time. To resolve this, we propose a novel solution by employing a variant of generative adversarial networks, which is trained to identify intrinsic properties of PDF and to generate evasive PDF malware with the minimum perturbation to the original PDF. Our solution successfully generated evasive PDF malware with a maximum number of 12 manipulation operations and found effective against ML-based classifiers and AntiVirus engines provided by VirusTotal.**

***Index Terms*—Adversarial examples (AEs), evading portable document format (PDF) classifiers, generative adversarial networks, PDF malware.**

Manuscript received March 17, 2021; revised June 23, 2021; accepted July 23, 2021. Date of publication August 12, 2021; date of current version October 13, 2021. This article was recommended for publication by Associate Editor Y. S. Ong upon evaluation of the reviewers' comments. This work was supported in part by the National Research Foundation of Korea (NRF) under Grant NRF-2020R1A2B5B03095204 and Grant 2018R1A2B3001628 funded by the Korea Government (Ministry of Science and ICT), in part by the BK21 FOUR Program of the Education and Research Program for Future ICT Pioneers, Seoul National University, in 2021, in part by the Inter-University Semiconductor Research Center, in part by the Institute for Information and Communications Technology Promotion under Grant 2020-0-01840 (Analysis on Technique of Accessing and Acquiring User Data in Smartphone) and Grant 2018-0-00230 (Development on Autonomous Trust Enhancement Technology of IoT Device and Study on Adaptive IoT Security Open Architecture based on Global Standardization—TrusThingz Project) funded by the Korea Government (Ministry of Science and ICT), and in part by the Ewha Womans University Research Grant of 2021. *(Ho Bae and Younghan Lee contributed equally to this work.) (Corresponding authors: Sungroh Yoon; Yunheung Paek.)*

Ho Bae is with the Department of Cyber Security, Ewha Womans University, Seoul 03760, South Korea (e-mail: hobae@ewha.ac.kr).

Younghan Lee, Yohan Kim, Uiwon Hwang, and Yunheung Paek are with the Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea (e-mail: 201younghanlee@snu.ac.kr; vkvkakal@snu.ac.kr; uiwon.hwang@snu.ac.kr; ypaek@snu.ac.kr).

Sungroh Yoon is with the Department of Electrical and Computer Engineering, Interdisciplinary Program in Bioinformatics, Interdisciplinary Program in Artificial Intelligence, Automation and Systems Research Institute, Institute of New Media and Communications, and Institute of Engineering Research, Seoul National University, Seoul 08826, South Korea (e-mail: sryoon@snu.ac.kr).

Digital Object Identifier 10.1109/TAI.2021.3103139

I. Introduction

MACHINE learning (ML) has extensively adapted in a large number of application areas, including speech recognition and image processing. One important such area is security, to which a variety of ML-based techniques have been applied in recent years. Several studies have empirically evinced a great potential and effectiveness of ML in solving certain security problems such as malware detection [1], [2]. In particular, the ML techniques for malware detection have been developed for years diverging to many different security problem domains, such as clustering of malware families [3], [4], detection of malicious downloads [5], [6], detection of account misuse networks [7], [8], detection of commonly exploited file formats (e.g., Java archives [9] and documents [10], [11]), and detection of portable document format (PDF) malware [12]–[15].

Not surprisingly, as ML becomes a dominant means for malware analysis, there is a growing temptation to find *adversarial examples* (AEs) that can diminish its effectiveness. Many latest

studies of AE attacks on ML [16]–[19] have demonstrated that a small perturbation to an input may forcibly change the prediction result of both ML and, newly surfacing, *deep learning* (DL) models. The studies suggest that the victim of AE attacks can be anyone from an entire spectrum of application areas where ML is applicable. As a result, this fact poses a daunting challenge to developers of ML models for malware detection [20]. Thus, it is crucial to build a robust malware detectors or classifiers that are resistant to AE attacks. One solution adopted by many in practice is to harden their model by training it with all possible AEs against it taken into account. For this, significant effort has been made to identify AEs against existing ML models for various malware detectors. In this article, we are interested in finding AEs that can be used to evade all the current (academic and commercial) ML-based PDF malware classifiers. Our interest originates from the fact [21] that as malicious PDF files have been known to be the most dangerous type of attack exploited by adversaries to date, ML-based techniques are being actively studied and developed to mitigate them even most recently.

Since its introduction, PDF has risen in great popularity and become the de facto standard for many different purposes of information sharing, such as text documents, data files, and presentation materials. PDF includes not only static content (e.g., texts and styles) but also dynamic content (e.g., JavaScript code and action triggers). The versatility of PDF files comes in their capability of displaying such rich content on virtually all kinds of today's computer systems and platforms. The popularity and versatility have been capitalized on by adversaries in a way that the PDF format files are used to craft diverse attacks on viewer applications, inflicting extensive damage on countless victims. One key attribute of PDF exploited by adversaries is its high connectivity to other objects, which facilitates modification of a PDF file, ultimately leading to an injection of a malicious load to the file. Another is the innate complexity of its file format, which facilitates malicious contents being concealed from the detectors. For example, JavaScript-based attacks deceive the detectors by injecting Javascript code in multiple objects at different locations inside the file. Such PDF malware is not only posing in the present but also likely to pose in the future, an immense threat to cybersecurity. It was reported by SonicWall [22], a private network security company, that more than 47 000 new attacks related to PDF files were discovered last year, and 73 000 PDF-based attacks were discovered in March, 2019 alone.

To prepare for the flooding of future zero-day PDF malware, much research has been done to improve the performance of PDF classifiers by employing ML techniques. As the first work in this direction, PDFrate-v1 [13] tackled the challenge with ML techniques by using metadata and *contents* of PDF documents. The approach characterized the documents' attributes by handcrafting 202 features to train a random forest (RF) model for detection. PDFrate-v2 [14] further improved the performance by taking advantage of an ensemble training technique. Hidost [12], [15] attempted to extract the files into a *structural* map and used it as a feature set for their train model. Support vector machines (SVMs) and RF are used as classification models, and both of them attain an impressive performance of detection.

As ML techniques for PDF classification become advanced and sophisticated, so do AE attack techniques for evading them. In principle, the purpose of these attack techniques is generating evasive PDF samples (i.e., AEs) against ML-based classifiers by picking and manipulating structural features that the classifiers utilize for detection. The early forms of AE attacks, which we collectively call *mimicry* attacks [23]–[25], aim to induce misclassifications of the classifiers by camouflaging malicious PDF files as benign ones. Unfortunately, mimicry attacks rely heavily on human expertise to understand a given malicious PDF file before finding fake structural features that will be added to the original file for aligning it with a known benign file. This implies that the success of their techniques would be strictly limited by human effort as well as their knowledge of a complex structure of the PDF file format.

Researchers endeavored to overcome the limitation by minimizing human involvement. They proposed the evasion techniques that can automate the process of generating evasive samples for AE attacks. EvadeML [26] introduced a stochastic approach based on *genetic programming*, which repeatedly performs feature manipulations based on a random mutation algorithm until an evasive yet malicious PDF sample is successfully obtained as output. While the output sample maintains the input's original maliciousness, it, unlike the input, is guaranteed to be evasive as it will induce a misclassification of PDF malware classifiers. EvadeML exhibited its effectiveness by producing evasive samples of all 500 PDF malware files selected from Contagio malware archive [27]. A later work, EvadeHC [28], claimed to achieve the same performance as EvadeML, that is, succeeding in generating evasive samples for all 500 PDF malware files from Contagio. Moreover, they assumed a more restricted realistic attack scenario, where the attacker will only be given a binary prediction score from the PDF malware classifiers.

Despite the impressive success in their automated evasive sample generation, our analysis has revealed that the existing evasion techniques consume an excessively large amount of time to obtain each sample. Although EvadeHC has made some effort to speed up its generation time by applying a hill-climbing method to the *random mutation* algorithm, their numbers still have much room for improvement. According to our observation, the main factor that increases the total time taken to generate an evasive sample is the inherent difficulty of maintaining the original maliciousness even after several trials of operations being carried out to manipulate structural features, which often result a crash. To explain this, consider the PDF malware that is not originally evasive when being given as input to the evasion techniques like EvadeML or EvadeHC. In order to generate an evasive sample as output from the malicious file, they must transform the original file structure by manipulating (i.e., inserting, deleting, and replacing) its structural features. Conceivably, it often occurs during the transformation that the original file loses its maliciousness if a certain feature manipulation happens to corrupt a file structure essential to maintain its maliciousness. Fundamental ingredients in all of these are feature-space models of attacks.

To elaborate limitations of current attacks, let hereby S denote a set of all structural features of our target PDF file, which can be manipulated to generate our evasive sample. We also define S', a subset of S, whose elements are *robust features* crucial to maintaining the target's maliciousness, by which we mean that the maliciousness might be corrupted by changing any of them. As briefly mentioned earlier, the existing evasion algorithms "randomly" select one feature after another from S and "mutate" the features until they obtain an evasive sample. Suppose that they select and mutate features from S' by chance. Then, as has been defined, it is likely that the maliciousness of our target file is lost by error. Upon recognizing the loss of maliciousness, the existing algorithms undo the mutation on the feature to restore the lost maliciousness and try to select another feature from S. We have observed in the existing techniques that they suffer from quite frequent trials and errors, each of which causes a waste of time, consequently all in all inducing a significant increase in the total time for sample generation.

A remedy for this problem would be to pinpoint the robust features and transform the input PDF malware by manipulating features only from the nonrobust features in search for an evasive malware sample (i.e., $S - S'$). Sadly, none of the existing techniques listed above attempt to have knowledge of S' when they generate evasive samples. Our observation on previous work motivated us to develop a new solution, where we drastically reduce the sample generation time by avoiding wasteful cycles of trials and errors during our PDF transformation. In our solution, we first strive to identify a set S' of structural features that are relevant to malicious behaviors of most PDF malware available today. Next, during the transformation phase, we continuously refer to the set in order to ensure that our algorithm should select candidates for mutation from the complementary set of S'.

Clearly, in order for our solution to work successfully, we must be able to determine the set S' for the PDF format files. To achieve this, we employ the *generative adversarial networks* (GANs), which, if appropriately trained, can learn to identify intrinsic properties (including structural features) of benign and malicious PDFs. The power of GANs that identifies the structural features belonging to S' comes from their innate characteristic, namely, the adversarial interaction between their two components, the generator and discriminator, by which S' is formed. To avoid the time-consuming repetition of trials and errors, the generator constructs evasive samples by modifying features only in $S - S'$. Many existing GANs usually employ a single discriminator, through which a modified sample very similar to the original can be generated. To generate a modified sample structurally very similar to the original, GANs select features in $S - S'$, thereby conserving the original's malicious behavior as a result. However, the generated sample must not only maintain the desired maliciousness but also evade the targeted malware classifiers. To satisfy both the requirements, we have introduced a GAN variant that employs a target PDF malware classifier as the second discriminator to manipulate features only from $S - S'$ during our evasive sample generation. To adjust the dependence level of these two cooperative discriminators, we use an additional parameter that controls the balance between them. Let alone its speed in finding evasive malware samples, our solution has another advantage that it can operate even under a realistic black-box attack scenario, in which the classification score revealed from the malware classifiers is in binary form (i.e., benign or malicious) rather than a continuous classification score.

We have evaluated our solution, PDF-GAN, against three PDF malware classifiers. First of all, we have found that it can generate evasive samples (without any crash) for all 500 unique PDF malware files selected from the Contagio archive. Our proposed model successfully evades the target PDF malware classifiers with the maximum number of 12 manipulating operations by 13 times faster than the previous approaches. In contrast, EvadeML required the maximum of 354 and 85 feature manipulating operations to complete the generation of evasive samples for PDFrate-v1 and Hidost '13, respectively. To further demonstrate the effectiveness of our approach, we include in our evasion seed all three types (e.g., JavaScript, ActionScript, and File Embedding) of PDF malware as known by CVE-2018-9958 [29], CVE-2013-2729 [30], and CVE-2010-3654 [31]. The analysis reveals that our evasive samples are all generated without any crash exhibiting the same malicious behaviors as the original malware with minimum modification. Last of all, unlike previous work, we evaluate the evasiveness of our generated malware samples against AntiVirus engines from VirusTotal.

II. Background

In this section, we describe the threat model and current state-of-the-art PDF malware classifiers. Also, we elaborate on recent evasion attacks for such classifiers and categorization of the attack method.

A. Threat Model

Depending on the different levels of knowledge held by an attacker, attack scenarios can be categorized into three different classes: white-box (i.e., perfect knowledge), gray-box (i.e., limited knowledge), and black-box (i.e., zero knowledge) [21], [32]. The less information available to an attacker, the closer the attack scenario is to black-box. The types of information that can be provided to an attacker are threefold: 1) the training dataset and its labels; 2) the feature set and the feature extraction algorithm of the classifier with its extracted feature types; and 3) the knowledge of the classification function and its hyperparameters.

In the black-box scenario, an attacker is provided with no information about the classifiers and in the gray-box scenario, only minimal knowledge (e.g., the feature representation) is provided. EvadeML constructs evasive samples against Hidost'13 and PDFrate-v1 under a black-box attack scenario. They assumed that the classification score was revealed in a real number with many query attempts. EvadeHC also operates under a similar scenario, but the main difference is that the classification score was given in the binary form.

Our approach, PDF-GAN, operates in the black-box and gray-box for ML-based classifiers and commercial AntiVirus Engines, respectively, with the same assumption that many submissions of files are allowed. Moreover, the classifiers only

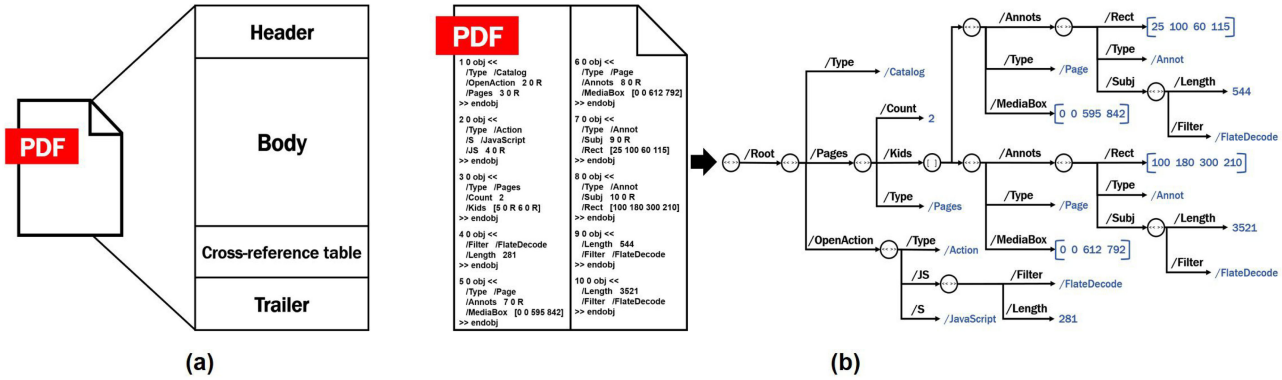

Fig. 1. (a) PDF structure. (b) Tree representation of PDF file, which is used as features in training PDF-GAN.

reveal the classification score in a binary form (i.e., benign or malicious). However, recently, many researchers managed to mitigate the evasion attack by limiting the number of queries as the current black-box attack requires many submissions. To this end, PDF-GAN was designed to also operate as a transfer-based attack [33]. For this, we trained a surrogate model that is a smaller network and evaluated the success rate of evasion with much fewer query attempts to the target classifiers. The details of the design and the experiments will be explained in the following sections.

B. Portable Document Format

1) PDF Structure: A PDF file can be broken down into four parts: *header*, *body*, *cross-reference table*, and *trailer*. Fig. 1(a) shows the structure of a PDF. A *header* contains rather simple information, which includes the version number of the PDF specification (i.e., "%PDF-1.3"). The *body* section contains objects and holds all data of the document. There are eight different types of objects supported by a PDF (i.e., Boolean, integer and real numbers, arrays, strings, dictionaries, names, streams, and null). A name object only contains unique values, whereas a dictionary object consists of a key and value pair.

Objects are identified by their given numbers, and they are either indirect objects or the direct objects constituting dictionaries. Indirect objects appear within the notation << >> and direct ones are denoted as follows:

$$6\ 0\ obj\ <<\ /Type/Action/S/JavaScript/JS\ 7\ 0\ R>>\ endobj$$

$$7\ 0\ \text{obj}\ <<\ /\text{Length}\ 231/\text{Filter}/\text{FlateDecode}\ >>$$

$$\text{stream}\ \cdots\ \text{endstream endobj}.$$

For example, object 6 with a keyword introduced by "/" will make an indirect jump to object 7, which contains a sequence of direct objects with keywords and their values. The length of the stream is 231, which requires a FlateDecode filter. A *cross-reference table* stores the mapping information of random and direct access, allowing a specific object to be found without having to search throughout the entire file. Note that PDF readers start rendering the PDF from the bottom of the file, which is the trailer. The *trailer* specifies the offset value for the PDF reader to find the cross-reference table and helps the reader find a specific object more quickly (i.e., trailer << /Size 9 /Root 1 0 R >> startxref 9178 %EOF). In this case, the offset is 9178 bytes, "/Size" indicates the number of entries in the cross-reference table, and "/Root" is the catalog dictionary for this file.

2) Types of PDF Malware: The three different types of PDF malware are briefly explained.

1) *JavaScript-based attacks* exploit a vulnerability using JavaScript code that can be embedded in one or several objects. Typical examples of such vulnerabilities are an API-based overflow and a Use-After-Free flaw.
2) *ActionScript-based attacks* capitalize on the fact that PDF files can visualize Flash content. This is usually achieved by embedding ShockWave Flash along with the ActionScript code such as memory corruption or corrupted file code.
3) *File-embedding attacks* take advantage of the fact that Adobe Reader can parse and read PDF files that are embedded with contents of different file types, such as images (e.g., bmp or tiff) and fonts (e.g., ttf). When reading a PDF file, embedded contents can lead to memory spraying to execute payloads with malicious activities.

C. PDF Malware Classifiers

1) Hidost: Hidost is implemented using two different types of classification models: SVM [12] and RF [15]. SVM is a supervised learning model that outputs an optimal hyperplane for separating two different labels. RF is a meta-estimator, comprising several decision trees that are merged for more accurate classification. As the first step, Hidost utilizes Poppler [34], a PDF parser, to dissect files into a structural multimap in its structure extraction stage. These structural paths of objects in a PDF are used as features during classification. Since there are many semantically equivalent yet syntactically different structures, a structural path consolidation, which is based on rules that are manually created, is carried out. For feature selection, Hidost naively includes only paths that are occurring in more than a certain number of files to form a feature set. Hidost is provided as open source, and the model was trained using 10 000 random

files with a malicious-to-benign ratio of 1:1. The entire PDF dataset was composed of 407 037 benign and 32 567 malicious files. The results indicated that the Hidost model was the top detection tool compared to AntiVirus engines (VirusTotal).

2) PDFrate-v2: The PDFrate classifier is implemented using an RF algorithm that applies an ensemble learning model designed to improve the prediction accuracy [14]. It employs *metadata* and the *content* of the PDF files as classification features, which include the names of the authors of the files, the size of the file, the position, and the number of specific keywords. The feature set is defined manually by the authors, and the total number of features is 202. However, only 135 are publicly available in the Mimicus implementation of PDFrate, which claims to achieve a close approximation. The main difference between PDFrate-v1 and PDFrate-v2 lies in the ML model applied. PDFrate-v2 adopts an ensemble method by applying mutual agreement among the classifiers. It introduces the idea of "uncertain" in the classifier votes, where rates of 25–50% are considered to be benign uncertainty, whereas rates of 50–75% imply malicious uncertainty. The effectiveness was tested against some known evasive attacks such as mimicry [25] and reverse mimicry [35], and impressive performance was demonstrated.

D. Evasion Attacks

1) Automatically Evading Classifiers: EvadeML presents a generic approach for evading the Hidost'13 and PDFrate-v1 classifiers through stochastic manipulations. It repeatedly mutates the original malicious PDFs to create evasive variants. It is an automated procedure, in which evasive samples manufactured by random mutations are tested by the oracle to check the presence of maliciousness. If no maliciousness is present, the variant will be returned to the mutation stage. As for the reliable malware signature, only the network behavior of the malware samples is considered. A total of 500 sample seeds were selected from the Contagio PDF malware dataset, and the proposed method successfully reached 100% evasion, which took approximately six days.

However, PDF-GAN is based on learning the difference in the feature sets between benign and malicious samples and modifying a malicious PDF with minimum effect on its original purpose, and hence achieving a 100% evasion success rate in noticeably less time and with fewer modifications.

2) Evading Classifiers by Morphing in the Dark: In this study, the authors focused on more restricted and realistic attack scenarios, where the target classifiers will only reveal the final prediction regarding whether they are benign or malicious. Hence, a scoring mechanism, EvadeHC, was proposed to overcome the limited information. The intuition behind this is to measure the number of steps to overturn the result of the detector and derive the real-value score from it. The authors introduced the notion of malice-flipping distance, which is the number of mutations required for a malicious PDF to lose its maliciousness as determined by a tester. The reject-flipping distance is a comparable concept, which is the number of morphing steps required for a malicious sample to be classified as benign. A simple morphing technique is employed that performs the basic operations: insert, delete, or replace. Their design consists of three components: a binary output detector, a tester to check the maliciousness of evasive samples, and a morpher that randomly mutates the PDF files. The target classifiers were Hidost'13 and PDFrate-v1, and its effects were evaluated with the 500 selected malware samples from Contagio archive.

Our approach operates under the strong assumption that the classifiers and testers only reveal their binary output results. Unlike this work, our PDF-GAN did not need a scoring function to convert the results into a real-value score and successfully evaded even more recent classifiers with the same seed samples.

E. Attack Method Categorization

1) Toward Adversarial Malware Detection: Lessons Learned From PDF-Based Attacks: This work provided a comprehensive taxonomy of various approaches of generating PDF malware and evaluated them from both methodological and experimental aspects [21]. The attacker's knowledge is explained in Section II-A. For the different types of evasion attacks on PDF malware classifiers, two main families have been introduced: optimization-based and heuristic-based. Optimization-based attacks attempt to formulate the attack as an optimization problem and heuristic-based attack, as the name suggests, attempts to *heuristically* make modifications in generating evasive PDF malware.

PDF-GAN can be categorized as an optimization-based evasion attack with both the black-box and gray-box scenario. Also, our solution successfully generates real samples by means of the repacking and verification process explained in Section III-F.

2) Intriguing Properties of Adversarial ML Attacks in the Problem Space: The authors proposed a formalization for evasion attacks in the problem space. The problem-space constraints consist of available transformation ($\mathcal{T}$), preserved semantics (Υ), robustness to preprocessing (Λ), and plausibility (Π). Our approach, PDF-GAN, can be summarized in these four formulations as follows.

$\mathcal{T}$: Addition/removal/change of elements in tree representation of PDF. Υ: Original malicious behavior (described in Section IV-D) is retained. Λ: Not robust to removal of features. Π: Generate AEs (i.e., evasive PDF malware) can be opened by a PDF reader.

III. Design

In this section, the design of our approach will be explained in detail. First, how features are extracted from PDF files and selected to be used as a feature set for training PDF-GAN. The explanation of how we selected a seed file for our mutation and evading model architecture will be followed by PDF repacking process. Fig. 2 shows an overview of our proposed method, which consists of three phases: 1) preprocessing of PDF; 2) PDF-GAN training; and 3) detection. The details are presented in the following sections.

A. Feature Extraction

PDFs are parsed into the tree representation, as shown in Fig. 1(b). We have utilized the PDFrw (i.e., PDF parser)

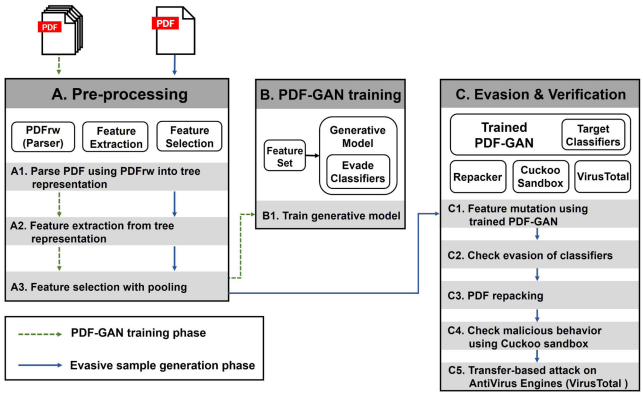

Fig. 2. Flowchart of the PDF-GAN framework.

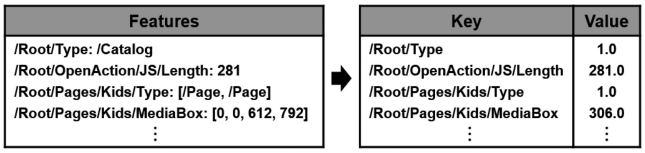

Fig. 3. Feature abstraction [Dictionary (Key:Value)].

provided by EvadeML [36] with few modification to correctly parse all objects. It is important that the parser do not omit any major objects (e.g., */Javascript* and */OpenAction*) as PDF-GAN would be unable to fully interpret the structural difference between benign and malicious PDFs, which, in turn, will lead to poor results in PDF-GAN's performance. Thus, to avoid leaving out any potentially pivotal information, PDFrw has been modified to include all key values of */Root* while parsing PDFs into a tree representation (i.e., */Metadata*, */OpenAction*, */Javascript*, */AcroForm*, */PageLayout*, etc.). Additionally, for higher speed computation, we ignore any paths containing an object with */Parent* or */Prev* as they are recursive.

From the tree representation, a feature set can be formed. Each path from the root to a leaf node and its value is considered as a feature, as listed in Fig. 3(left). The feature abstraction is performed by converting features into a form of dictionaries (i.e., keys and values). Finally, similar to the previous work [12], [15], any values in a string type were converted to an integer value of 1 and a value given in the form of an array of values was transformed into the median of values in an array, as shown in Fig. 3(right).

B. Feature Selection Process

The feature selection process plays a crucial role in determining the effectiveness of the ML. Since it is impractical to use all features extracted from the entire dataset, we must select a decent number of features that represent all features in a sufficient manner, to be included in the feature set. In previous studies, Hidost simply narrowed down the size of the feature set by only including features that occur in more than a certain number of files. Such a number is typically set to 1% of the training set size and the selection is performed "in hindsight"

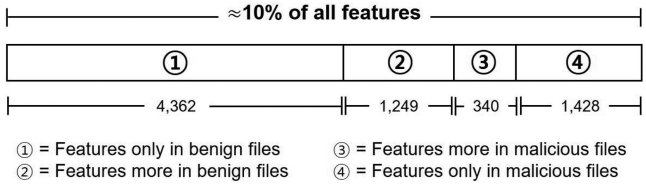

Fig. 4. Feature selection by pooling.

TABLE I
MUTATION SEED SELECTION

Description	No. of seed
Contagio dataset (excl. training set)	6,105
Cuckoo result with network activities	1,503
PDFrw parsing	1,502
Feature extraction	1,485
Unique files	712
True positive of SVM & RF & Ensemble	709
Randomly selected samples	500

once for the entire dataset. However, this method failed to evenly include features from both benign and malicious files as malicious files tend to contain unique features. Hence, a huge portion of the feature set was occupied by features extracted from benign PDFs. Therefore, with the intention of including features extracted from malicious files, a novel feature selection process was applied. In short, the entire feature set was created by deliberately maneuvering the ratio between different pools of features to be used for training the target classifiers and PDF-GAN.

We segregated the entire features into four pools: 1) features found only in benign PDFs; 2) features found more in benign PDFs than malicious ones; 3) features found more in malicious PDFs than benign ones; and 4) features found only in malicious PDFs. Fig. 4 shows the number of features from each pool. Any features that were found in less than that of either benign or malicious files were excluded. The main reason for applying the new feature selection process was to overcome the imbalance of dataset problem commonly found in DL that the existing mechanism failed to overcome, as explained above. We gathered a set of 297 features with a balanced number of features from each pool. This method resulted in an improved detection performance, as it will be illustrated in Section IV-B.

C. Seed Selection for Mutation

We have selected a total of 500 files after filtering out from Contagio malicious files. Table I shows how those files were chosen. Selecting seed PDF files to be fed into PDF-GAN was done with the dynamic analysis system (i.e., Cuckoo: the leading open-source automated malware analysis system). First, a primitive set of seeds was selected from the Contagio dataset, excluding files from the training set. After analyzing 6105 files, it appeared that only 1503 files indicated some malicious network activities. This result may have been caused by the inborn limitation of dynamic analysis. All of these files were parsed through PDFrw and the feature extraction process to

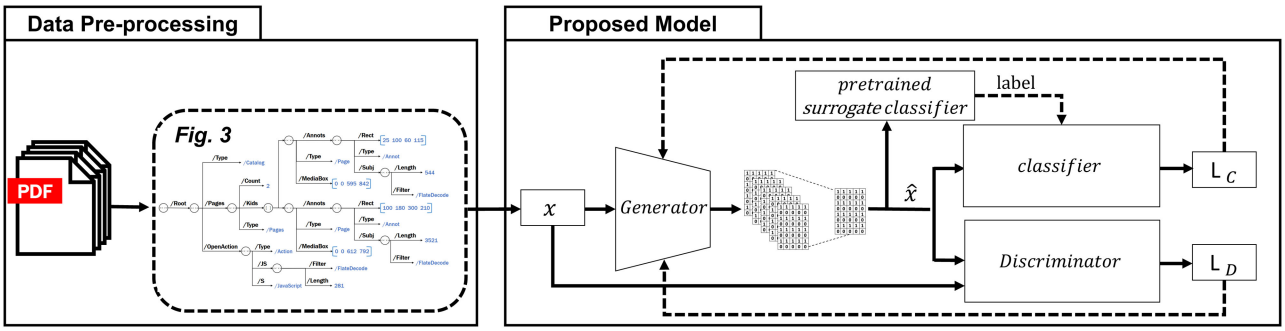

Fig. 5. Model architecture. In the data preprocessing stage, PDF is parsed into tree representation and used as a feature for training the proposed model. The proposed model is described in detail in Section III-D.

validate their tree structure. A total of 1485 files remained after this process. Upon close examination, we realized that many of the files shared the exact same value for the set of 297 features. Therefore, after filtering out homogeneous files, 712 remained. Finally, it was important that these files be classified as malicious by our target classifiers. The remaining files were put through all three classifiers sequentially and 99.6% of them were corrected classified as malicious, which demonstrates the high performance of our classifiers. Among 709 files, we randomly selected 500 for the evasion process to evaluate against previous studies.

D. Evading Model

Our training involved two types of adversarial game: an adversarial game for mimicry and an adversarial game for evasion. The model consists of four parts: a generator, a discriminator, a (pretrained) surrogate classifier, and an adversarial classifier. The generator constructs a PDF close to the form of the input data, and the discriminator then predicts a confidence score on whether the generated data are close to the form of original input data. The surrogate classifier produces a prediction score of the generated data, and the outputs are used to train the classifier. The classifier adopts to collateral learning by adversarial training, making the classifier robust against unknown features. The generator is trained with the original PDF to learn and create a variant version of the original PDF such that the prediction result of the generated data is a reverse of the original PDF.

The architecture of this model is illustrated in Fig. 5. The generator takes an input x and gives an output $\hat{x}$, both of which are given to the discriminator and the classifiers. The discriminator outputs the probability that $x = \hat{x}$, and the classifier outputs a prediction score given input $\hat{x}$. The learning objective of the generator is to minimize the prediction score of the classifier, and the learning objective of the discriminator is to discriminate whether a generated PDF is the original x. The generator, G, aims to generate malicious PDF by learning data distribution close to the distribution of benign PDF. For each malicious input $x \in X$, G seeks a possible stochastic mapping to other representation, $\hat{x} = G(x; \theta_G) \in \hat{X}$ by conditional probability density function $p(\hat{x}|x)$, where θ_G denotes the parameter for the generator. In original GANs, the generator receives noise $z \sim p_z(z|\mathbb{Y})$, where $\mathbb{Y}$ is the class labels space. However, in our model, G receives noise z, which is computed by $x \times r \in \mathbb{R}^d$, where r and d are the random string and the feature dimension, respectively. The discriminator, D, aims to distinguish malicious features by learning distribution of S', and the generator's input is required to retain the original form of maliciousness by computing reconstruction loss between generator's input and the output. The surrogate classifier, C^s, aims to train the classifier by predicting prediction score given generator's output. The classifier, C, aims to evade the target classifier by adversarial training given generator's output and its prediction score. Computed loss from both two discriminators (discriminator and classifier) are then backpropagated to the next step training step of the generator. For the white-box attack, C^s can be replaced by the actual target classifier.

To define the learning objective, let L_G, L_D, L_{C^s}, and L_C denote the loss of the generator, discriminator, surrogate classifier, and classifier, respectively. The generator then has the following objective functions:

$$\begin{aligned} \mathrm{L}_G &= d\left(x, \frac{1}{n}\sum_{i=0}^{n}(G(x,\theta_G)_i)\right) + ((1-\lambda_d)\cdot \mathrm{L}_D + \lambda_d \cdot \mathrm{L}_C) \\ &= d(x,\hat{x}) + ((1-\lambda_d)\cdot \mathrm{L}_D + \lambda_d \cdot \mathrm{L}_C) \end{aligned} \tag{1}$$

where $d(x, \hat{x})$ is the Euclidean distance between the original and arithmetic mean of generated data. In addition, λ_d is the weight parameter for the surrogate classifier that can be used to control the dependence level of the generation to maximize the diversity of the feature changes.

A discriminator has the sigmoid cross entropy loss of

$$\begin{aligned} \mathrm{L}_D = & -y \cdot \log(D(\psi(\hat{x}) \cdot \psi(x)) \\ & -(1-y)\cdot \log\left(1 - D\left(\frac{1}{n}\sum_{i=0}^{n}(G(x,\theta_G)_i)\right)\right) \end{aligned} \tag{2}$$

where ψ is the binary Hamming distance between $\hat{x}$ and x.

A classifier has an objective function of

$$\mathrm{L}_C = -||C(x) - C^s(\hat{x})||_2. \tag{3}$$

In the case where input x is classified as malicious by the surrogate classifier, $\hat{x}$ needs to be classified as benign. For this,

we need to maximize the distance of the prediction score. Given the above equations, the generator optimizes a convex of (1) finding Nash equilibrium of a minmax game between the G against both D and C.

A surrogate classifier has an objective function of

$$\mathrm{L}_{C^s} = (y - f(x))^2 \tag{4}$$

where f is a simple 1-D convolution neural networks with one hidden layer.

E. Model Architecture

For GAN architecture, we optimized the architecture to accommodate the PDF malware domain by following a similar concept of visual representation learning. The generator of the first layer consists of 64 filters with a length of 4, the second layer consists of 32 filters with a length of 2, and the third layer consists of 16 filters with a length of 2. The fourth layer consists of eight filters with a length of 2. Additional layers are then added in reverse order as the number of input length n. Batch normalization [37] is used at each layer and tanh [38] is used as the activation function at each layer, except for the final layer where rectified linear unit [39] is used for the activation function. A sigmoid activation function is used to output the probabilities from the logits. The discrimination and the classifier of the first layer comprise n input feature size of filters; second $n \times 2$; third $n \times 4$; fourth $n \times 8$; and fifth n input feature size of filters. Tanh is used at each layer as the activation, except for the final layer, where a sigmoid activation function is used to output probabilities from the logits. For the pretrained surrogate classifier, we constructed one layer network. The kernel size of each layer is 3 with stride 1 for all networks, and only 10% of the training dataset is used with 100 epochs.

F. PDF Repacking and Verification

Once PDF-GAN successfully evaded the classifiers with the mutated feature set, we must verify that the originally intended maliciousness was retained. For this verification, the mutated features must be applied to the original malicious PDF by the repacker. The repacker has three operations: *insert*, *replace*, and *delete*. *insert* operation updates the dictionary according to the mutated feature set. A new key and value is inserted into PDF, and the new value was in the form of real numeric value. For example, */Root/Pages/Rotate: None* $\longrightarrow 0$ means that the new path of */Root/Pages/Rotate* with a value of 0 is inserted into the tree representation. *replace* and *delete* operations are essentially operate in a similar manner. The former replaces the existing value with the new value, and the latter deletes the key and the value pair (i.e., feature) from the dictionary, hence deleting a path from the tree representation. All three operations are carried out while retaining the tree representation structure of PDF.

The most time-consuming aspect of finding evasive samples is repacking the mutated features back into proper PDF files. In addition to GAN reconstruction power, addition trick was considered to reduce the number of tries in repacking to reconstruct the PDF files. As explained in Section III-A, if the dictionary was in an array from, the median value of elements was used instead. Therefore, in the repacking process, the modified feature value, which is a form of real numeric value, was reshaped into an original form (e.g., an array form). Consequently, it contributed in improving the evasion speed compared to the state-of-the-art evasion techniques, which is described in Section V-D.

The reconstructed PDF file (i.e., repacked evasive sample) is tested in Cuckoo sandbox to verify that the maliciousness is maintained. The detail of this verification stage is explained in Section IV-D.

IV. Experiment

For the experimental evaluation, a Cuckoo Sandbox 2.0.7 was arranged using 16 virtual machines (VMs) running Windows XP 32 bit SP3 and Windows 10. Adobe Reader 8.1.1, PDF-XChange 2.5, and Foxit Reader 9.0.1 were installed in VMs. For the training, we used a Linux machine [Ubuntu 12.04, 2.6 GHz Intel Xeon E5-2690 (14 Cores) and 1 NVIDIA GTX TITAN V 12GB] without parallelization.

A. Datasets and Model Training

We managed to gather PDF malware samples from VirusTotal. The dataset was collected on December 20, 2017 and on March 14, June 19, and July 17, 2018, and it consisted of 10 673 files in total. The Contagio dataset comprises a total of 9109 benign files and 11 105 malicious files. In addition, common vulnerabilities and exposure (CVE) samples were collected from Exploit-db [45], where proof-of-concept codes and files are uploaded. Six specific samples were used in the experiment.

For training PDF-GAN, we used an optimizer of the multi-class logarithmic loss function Adam [46] with a learning rate of 0.001, a beta rate of 0.5, and a minibatch size of 16. The discriminator achieved optimal loss after 1000 steps, whereas the generator required 1500 steps to generate original data similar to the sample. Most of these parameters and network structures were experimentally determined to achieve optimal performance. Randomly selected 5000 benign and 5000 malicious files from the Contagio dataset were used for training classifiers and PDF-GAN.

B. Target Classifiers

Our target classifiers were Hidost'13 with an SVM model and Hidost'16 with an RF model, both of which used Poppler as a parser and PDFrate-v2 with an ensemble method that used a customized parser. The feature extraction and selection process were reproduced according to an open-source code, and each ML model was applied using scikit-learn. Well-known detection performance parameters such as accuracy, $F1$-score, and the area under the curve (AUC) were measured.

For Hidost, the test set comprised the Contagio dataset excluding the samples used for the training (i.e., 6105 files) and PDF malware samples from VirustTotal (i.e., 10 673 files) that were not included in the training set also. For PDFrate-v2, we applied the same methodology as explained by the authors, and the classifier was applied to the training set using tenfold cross validation. The results are presented in Table II under *Old*.

TABLE II
DETECTION ACCURACY OF PDF-GAN COMPARED TO TARGET CLASSIFIERS

	SVM (Hidost 13')		RF (Hidost 16')		Ensemble (PDFrate-v2)	
Feature Selection	Original	**New**	Original	**New**	Original	**New**
Accuracy	96.46%	**96.89%**	96.45%	**98.07%**	99.37%	**99.69%**
AUC	0.9886	**0.9936**	0.9880	**0.9936**	0.9932	**0.9948**

TABLE III
DETAILS OF CVES USED IN THE EXPERIMENT

CVE-ID	Type of PDF malware	Type of vulnerability	Application	Version	Exploited (Arbitrary code execution)
CVE-2008-2992 [41]	JavaScript	Buffer overflow	Adobe Acrobat & Reader	8.1.2	calc.exe & notepad.exe & message-box
CVE-2010-0188 [42]	File Embedding (.tiff)	Integer overflow	Adobe Acrobat & Reader	9.1	calc.exe
CVE-2010-2883 [43]	ActionScript	Buffer overflow (CoolType.dll)	Adobe Acrobat & Reader	9.3.4	calc.exe & notepad.exe
CVE-2010-3654 [31]	ActionScript	Flash (Memory corruption)	Adobe Acrobat & Reader	9.4	calc.exe
CVE-2011-2462 [44]	JavaScript	Flash (Buffer overflow)	Adobe Acrobat & Reader	9.4	calc.exe & message-box
CVE-2013-2729 [30]	File Embedding (.bmp)	Integer overflow	Adobe Acrobat & Reader	10.1.4	message-box
CVE-2017-13056 [45]	JavaScript	Improper validation of string	PDF-XChange	2.5	calc.exe
CVE-2018-9958 [29]	JavaScript	Use-After-Free	Foxit Reader	9.0.1	calc.exe

However, with the help of our unique feature selection process, we were able to achieve an even better detection performance as shown under the heading *New* in Table II. The performance was measured with all varying factors including the training and test datasets fixed except for the feature selection process and the performance showed noticeable improvement across all fields. Hence, we can claim that, because our own classifiers performed better in terms of accuracy, AUC, and $F1$-score, it is reasonable to target the new classifiers instead of Hidost'13, Hidost'16, and PDFrate-v2.

C. CVEs for Various Types of PDF Malware

CVEs provide publicly known security vulnerabilities in a reference style. To widen the scope of capability in terms of evasion effectiveness, the CVEs described in Table III were included in the experiment in generating evasive samples. The set comprises three types of PDF malware, namely, JavaScript, ActionScript, and File embedding. The types of vulnerability also varied widely from a buffer overflow to memory corruption and Use-After-Free. In addition, several different target applications were tested including Adobe Acrobat & Reader and Foxit Reader. Adobe versions required for the successful attack range from 8.1.2 to 10.1.4. We implemented the attack using a few different codes to be executed when the PDF is parsed and viewed with the reader. It is important to note that these samples were not included in the training phase of the framework, but only after PDF-GAN was fully trained, these samples were fed into a trained PDF-GAN model for the purposes of generating evasive samples.

D. Malicious Signature

Maintaining the maliciousness of PDF files was an absolute necessity in confirming the evasive sample and completing the evasion of the classifiers. Had they lost maliciousness at any stage of the evasion process, the purpose of this article would have been negated. To check if mutated malicious PDF files still acted with malevolence, we leveraged Cuckoo sandbox. Cuckoo can analyze many different malicious files and trace API calls and the general behavior of files transformed into comprehensible signatures. Owing to the innate limitations of a dynamic analysis, the behavioral signatures varied even for the same file. Therefore, reliable malicious signatures were needed to confirm that the modified PDFs still maintained their maliciousness. There were three main types of analysis we paid special attention to: network, behavior, and static. Table IV shows the types of signatures and their examples. We compared the analysis results between the original and modified versions. However, focusing only on the network behavior of the files would limit our work to malware related to network activities. Hence, unlike previous work, CVEs of all three types of PDF malware described in Section II-B2 were included in the experiment. After the modifications were made using trained PDF-GAN, the modified file was put through a tester stage. If it performed as originally designed, as listed in Table III, it was considered to be an evasive sample.

E. AntiVirus Engines (VirusTotal)

VirusTotal investigates submitted URLs or files with AntiVirus engines and reveals the detection result from each engine. Although PDF-GAN already proved its imposing capability in generating AEs by evading open-source PDF malware classifiers, we further demonstrate its effectiveness by evading commercial AntiVirus engines. Tested AntiVirus engine version was the most recent update (i.e., 2020. April). The procedure for this attack consists of two stages: 1) use generated AEs from the Contagio dataset to check if any of them can evade AntiVirus engines (i.e., a transfer-based attack); and 2) generate variants of successful AEs to further improve the evasion rate for more AntiVirus engines. The result of stage 1 is illustrated in Section V-E, and it shows that many AEs appeared to be also effective on numerous AntiVirus engines through a transfer-based attack (i.e., PDF-GAN is not trained to evade any of

TABLE IV
MALICIOUS SIGNATURES

Analysis Type	Description	Example
Network	Performs some HTTP requests	[GET http://www.deaf-video.de/3c55ea9320fcadfabb79d08f91bef510/.a1/load.php?e=2]
	DNS queries	[www.deaf-video.de]
	Transport IP addresses (UDP, TCP)	[UDP: 192.168.56.128:137 –> 192.168.56.1:137] [TCP:192.168.56.128:1292 –> 185.53.178.6:80]
	Network communications indicative of a potential or script payload download (API: URLDownloadToFileW)	[url: http://www.deaf-video.de/3c55ea9320fcadfabb79d08f91bef510/.a1/load.php?e=2] [filepath_r: C:\DOCUME 1\cuckoo\LOCALS 1\Temp\e.exe]
Behavior	One or more non-whitelisted processes were created	[C:\DOCUME 1\cuckoo\LOCALS 1\Temp\e.exe]
Static	The PDF open action contains JavaScript code	[<< /S /JavaScript /JS this.BXcfTYewQ() >>]

TABLE V
FEATURE MUTATION RESULT FOR THE CONTAGIO DATASET AND CVES (N = NUMBER OF FILE AFFECTED BY THE OPERATION)

Features	Contagio dataset						CVE-IDs					
	SVM		Random Forest		Ensemble		SVM		Random Forest		Ensemble	
	Operation	N	Operation	N	Operation	N	Operation	N	Operation	N	Operation	N
/Root/Type	['Insert', 'Change']	500	['Insert', 'Change']	500	['Insert', 'Change']	500	['Change']	14	['Change']	14	['Change']	14
/Root/Pages/Type	['Change']	500	['Change']	500	['Change']	500	['Change']	14	['Change']	14	['Change']	14
/Root/Pages/Rotate	['Insert']	500	['Insert']	500	['Insert']	500	['Insert']	14	['Insert']	14	['Insert']	14
/Root/Pages/Kids/Type	['Insert', 'Change']	500	['Insert', 'Change']	500	['Insert', 'Change']	500	['Change']	14	['Change']	14	['Change']	14
/Root/Pages/Kids/Resources/ProcSet	['Insert', 'Change']	396	['Insert', 'Change']	418	['Insert', 'Change']	442	['Insert', 'Change']	10	['Insert', 'Change']	13	['Insert', 'Change']	11
/Root/Pages/Kids/MediaBox	['Insert', 'Change']	269	['Change']	268	['Change']	268	['Change']	2	['Change']	5	['Change']	3
/Root/Pages/Kids/TrimBox	['Change']	234	['Change']	234	['Change']	234	-	-	-	-	-	-
/Root/Pages/Kids/Contents/Filter	['Insert', 'Change']	220	['Insert', 'Change']	204	['Insert', 'Change']	289	['Insert', 'Change']	5	['Insert', 'Change']	12	['Insert', 'Change']	6
/Root/Pages/MediaBox	['Change']	73	['Change']	73	['Change']	73	['Change']	2	['Change']	2	['Change']	2
/Root/Pages/Kids/CropBox	['Change']	18	['Change']	18	['Change']	18	-	-	-	-	-	-
/Root/Pages/Kids/BleedBox	['Change']	18	['Change']	18	['Change']	18	-	-	-	-	-	-
/Root/Pages/Kids/ArtBox	['Change']	18	['Change']	18	['Change']	18	-	-	-	-	-	-
/Root/Names/JavaScript/Names/JS/Length	['Change']	17	['Change']	17	['Change']	17	-	-	-	-	-	-
/Root/AcroForm/Fields/Kids/Kids/Rect	['Change']	8	['Change']	8	['Change']	8	-	-	-	-	-	-
/Root/Pages/Kids/Contents/Length	['Change']	7	['Change']	7	['Change']	7	-	-	-	-	-	-
/Root/Pages/Kids/Annots/Rect	['Change']	4	['Change']	4	['Change']	4	['Change']	2	['Change']	2	['Change']	2
/Root/OpenAction/Annots/Rect	['Change']	3	['Change']	3	['Change']	3	-	-	-	-	-	-
/Root/Pages/Kids/Annots/Subj/Length	['Change']	1	['Change']	1	['Change']	1	-	-	-	-	-	-

AntiVirus engines). The variants of AEs were created by swapping malicious contents from other malware samples among detected as malicious by engines. This approach is rooted from the understanding that PDF-GAN successfully discovered PDF structure that can evade some engines, and by only changing the contents, more AEs can be generated. The AntiVirus analysis result for 45 engines is shown in Section V-E.

V. RESULTS

In this section, the experimental result for evading PDF malware classifiers and maintaining maliciousness using both Contagio dataset and CVEs is described. Also, there was an drastic improvement in the time required to evade PDF malware classifiers for 500 samples. Finally, the result of an experiment to evade AntiVirus engines (Virustotal) is explained.

A. Feature Mutation Result for Contagio

The feature mutation results are shown in Table V. All of 18 features that were manipulated in at least one file were from the set $S - S'$ as the desired maliciousness was maintained. There were four features listed at the top that needed to be altered in all of 500 files to evade the classifiers. In addition, none of the features were removed from the original files. We believe that this fact may have been crucial in retaining the maliciousness and passing the Cuckoo test phase on the first attempt. If any of the features from a set of features relevant to the malicious behavior (i.e., S') were modified, many malware files would have lost their original maliciousness.

As mentioned in Section III-A, a value assigned as an integer value of 1 indicates that it was initially in a string type. Hence, if such a value is changed to any other value, the original meaning will be diminished. There were five cases in which the value was changed from 1 to 2

$$\{/Root/Type,\ /Root/Pages/Type,\ /Root/Pages/Kids/Type,\ /Root/Pages/Kids/Contents/Filter,\ /Root/Pages/Kids/Resources/ProcSet\}.$$

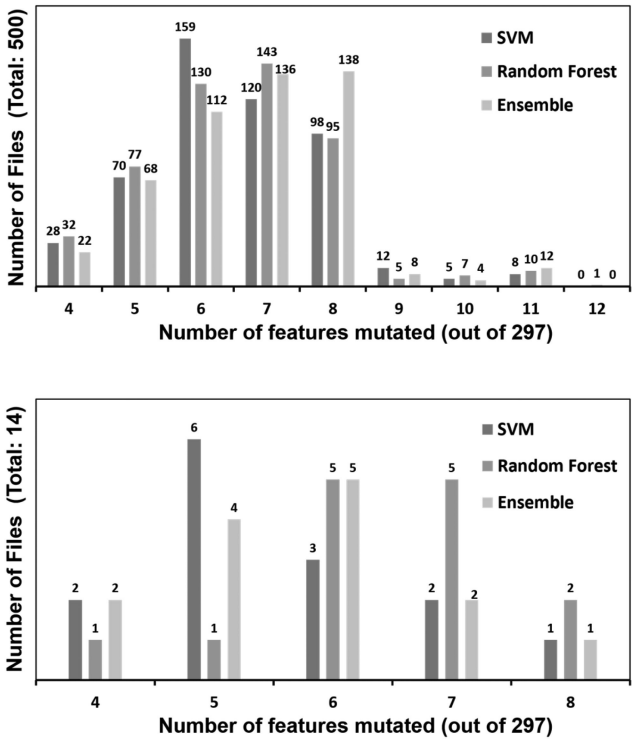

Fig. 6. Number of features mutated to generate AEs for all 500 files selected from Contagio (top) and 14 CVE files (bottom).

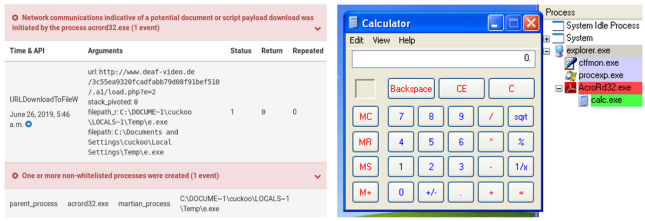

Fig. 7. Cuckoo signature result (left), arbitrary code execution result of CVE-2011-2462 (right).

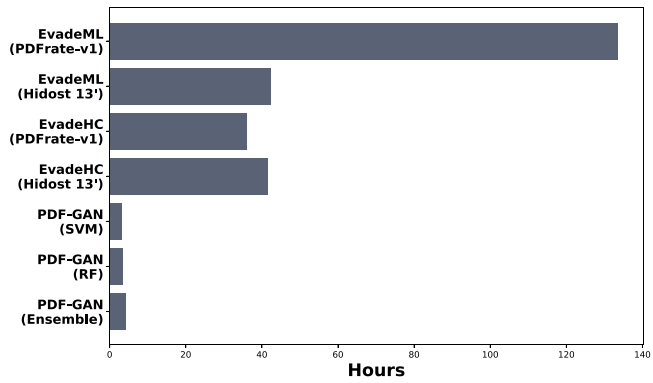

Fig. 8. Time required to evade PDF malware classifiers for 500 selected malware files from Contagio.

Another interesting observation is that to evade all classifiers, *insert* operation on */Root/Pages/Rotate* was required.

Our approach, PDF-GAN, unlike EvadeML and EvadeHC, grasps the differences in the patterns of the features between benign and malicious files and opts only to modify the minimum number of features from the files. The number of mutations required for the files differed between learning algorithms, as shown in Fig. 6(top). Fewer than eight features were needed to be modified in more than 95% of the files.

To confirm that PDF-GAN truly deduced the distinction in the patterns of the features to incur the minimum number of feature manipulations, we partially modified the file according to PDF-GAN. For example, for a file that required five features to be modified, 31 partially modified variants were created. (i.e., 5 combination (C) 1 + 5C2 + 5C3 + 5C4+ 5C5 = 31). The result clearly showed that PDF-GAN provided the least number of modifications to complete finding evasive examples. For the case of PDFrate-v2 (Ensemble), generating evasive examples for all 500 original PDF malware at the point in which all features suggested by PDF-GAN were altered. Therefore, we can conclude that the our approach suggested all the features that were needed to be perturbed in order to evade the classifiers.

B. Feature Mutation Result for CVEs

Not surprisingly, the result of feature mutation for the CVE samples showed a significant similarity compared to that of the Contagio samples. Table V illustrates that all the modified features were among those in the results of the Contagio samples. The top four features that affected the entire Contagio samples were also affected by all 14 CVE samples. Also, no feature was deleted from the original malware. On top of this, the number of mutations requires to find evasive samples shows the same trend as illustrated in Fig. 6(bottom). This result confirms that PDF-GAN was trained to alter only those features that deceive the classifiers while preserving the intended maliciousness.

C. Malicious Signature Verification

The result of preserving a malicious signature was confirmed by Cuckoo and by manually running the CVE samples. An example of the results is shown in Fig. 7. By analyzing the Cuckoo results, we confirmed that all signatures listed in Table IV remained intact for all 500 seed samples, which verified the successful attempt of generating evasive samples. Moreover, all of the CVEs that contained all three types of PDF malware also operated according to the initial intention of the malware. The right side of Fig. 7 shows that a calculator opened when the malicious PDF was read by Adobe Reader. The attacker can customize the exploit to have any code executed at will.

D. Evasion Speed

As our approach tackles the problem by training PDF-GAN with the feature set, it was expected that the cost of execution in terms of evasion speed would be better than that of the previous work by EvadeML. Fig. 8 illustrates the total time taken to identify an evasive variant for all 500 selected malware seeds. EvadeML employs a stochastic search based on a fitness function

meaning that many possible variants are created to be tested on the oracle for their malicious signature. As the unit cost of the Cuckoo sandbox testing was much higher than any other stages of the evasion, a huge portion of time spent by EvadeML was designated for oracle testing. However, our approach managed to avoid such overhead by utilizing PDF-GAN only to modify the nonrelevant features of PDFs in maintaining malicious behavior. Thus, we needed to perform only the verification stage once to confirm that all variants maintained their malicious signature.

All stages of our evasion process are shown in Fig. 8, including parsing of files, feature extraction, feature selection, training PDF-GAN, inference, and finally testing the possible evasive sample with the Cuckoo sandbox. As expected, the stages that occupied the largest portion were the training and inference stages with PDF-GAN. A total of 102 min was spent on the classifier with the SVM model (e.g., Hidost'13), 127 min on the classifier with the RF model (e.g., Hidost'16), and 180 min on the classifier with the ensemble model (e.g., PDFrate-v2). They correspond to 55%, 61%, and 69%, respectively, of the total time taken, respectively. The more robust the detector, the longer it took for PDF-GAN to be trained and to infer the modified version of PDFs.

In comparison to the total time taken for EvadeML to evade Hidost'13, our approach achieved the 100% evasion rate within about 3 h, which is more than 13 times faster in evading an SVM model. Moreover, even when PDF-GAN aimed to evade a more advanced classifier with the ensemble model, it achieved the full evasion 30 times faster than EvadML targeting PDFrate-v1, which is merely an RF model. In addition, we compared the time required for full evasion against the EvadeHC algorithm. The total evasion time against Hidost'13 was measured according to the author explaining that the average time taken to generate a single evasive sample was 5 min. In relation to PDFrate-v1, the relative time taken was measured for evading all 500 seed samples. Surprisingly, a contrast to EvadeML, the time taken to generate all evasive samples for Hidost was greater than that of PDFrate-v1 with the EvadeHC algorithm. In comparison to our PDF-GAN model, it managed to achieve the full evasion more than 13 times faster for an SVM model. It is important to note that while in our experiment setup, only 16 VMs were implemented, EvadeHC utilized 216 VMs. This implies that PDF-GAN could achieve the full evasion in a much shorter time if the same number of VMs were used for the testing phase.

E. AntiVirus Engines (VirusTotal) Result

All 500 malware samples selected for previous experiments and AEs that PDF-GAN generated to evade three classifiers were uploaded to VirusTotal for analysis from AntiVirus engines. The result is summarized in Table VI, and it shows the number of malware files detected and AEs for each engine. It is important to notice that AEs discovered for each engine are from the AEs that PDF-GAN generated while evading Hidost'13, Hidost'16, and PDFrate-v2. We observed that generated AEs were still effective for the AntiVirus engines (i.e., a transfer-based attack), and a total of 19 engines appeared to be vulnerable to this transfer-based attack.

TABLE VI
NUMBER OF MALWARE FILES DETECTED (OUT OF 500) BY ANTIVIRUS ENGINES AND AES BY A TRANSFER-BASED ATTACK

AntiVirus engines (VirusTotal)	No. of original files detected	No. of AEs
AegisLab	472	54
Arcabit	445	135
Avira	500	48
BitDefender	495	3
Comodo	472	49
DrWeb	279	1
ESET-NOD32	471	109
Emsisoft	492	1
F-Prot	483	4
F-Secure	495	78
GData	498	3
K7GW	138	5
McAfee-GW-Edition	498	1
MicroWorld-eScan	497	1
Microsoft	457	13
Sangfor	304	1
SentinelOne	500	2
TotalDefense	344	42
ViRobot	496	429

Furthermore, we created the variants of those AEs by swapping malicious contents from malware files, and Fig. 9 illustrates the successful evasion rate in 45 AntiVirus engines. Few engines were excluded as they did not support analysis for PDF format malware. As all 500 selected malware samples contain unique maliciousness, we defined that finding 500 AEs which collectively contains those maliciousness represents 100% evasion rate. 100% evasion rate was achieved in seven AntiVirus engines and 60% for 45 engines on average. These results showed that if the experiment did not rely on a transfer-based attack (i.e., if PDF-GAN is trained to evade AntiVirus engines), even higher evasion rates can be achieved.

VI. DISCUSSION

While PDF-GAN is effective under the black-box assumption that reflects its effectiveness, one can design a defensive mechanism for a more robust detection system. Similar to any other evasive techniques, multiple submissions to the detector are required to acquire a classification score. Therefore, if the defender decided to limit the number of submissions for a single peer, it would hinder the performance of the evasive technique. Moreover, the defender can opt to retrain the detector model with newly submitted files. Using newly submitted files, the detector model can be retrained and can employ recent ML approaches for continual learning [47]–[50], in which ML is used to continuously learn without loss of acquired knowledge on previous tasks.

In a strong black-box scenario where the number of queries is limited, it is imperative that PDF-GAN still shows a promising performance with an extremely small number of queries to the classifier. With a single-layer surrogate model, we managed to achieve 9.6% transfer rates. This is an improvement from other query limited black-box attacks, which showed 3.4% transfer

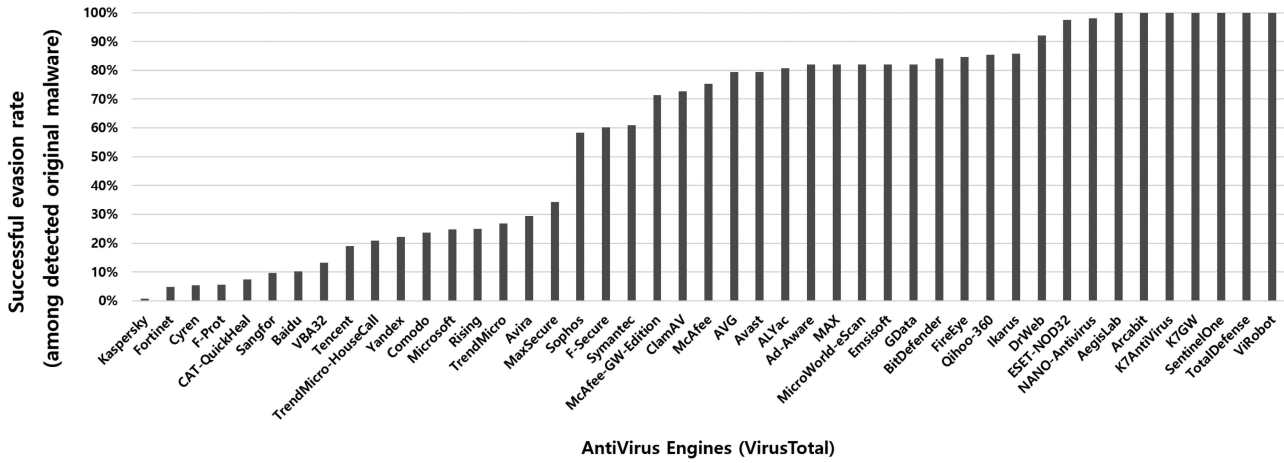

Fig. 9. Evasion rate of AntiVirus engines by generating variants of AEs in Table VI.

TABLE VII
DIFFERENT TYPES OF ADVERSARIAL ATTACKS ON ML MODELS

Type of Adversarial attacks	Fast Gradient Method (FGM)	Projected Gradient Descent (PGD)	Basic Iterative Method (BIM)	Carlini&Wagner (CW)
Is classifier evaded?	✓	✓	✓	✗
Is malicious signature maintained?	✗	✗	✗	✗
Is PDF repacking successful?	✗	✗	✗	✓
Is AntiVirus engines (VirusTotal) evaded?	✗	✗	✗	✗
Avg. No. of features mutated	126	120	121	29

rates [51]. If more queries are allowed to the target classifier, PDF-GAN can immensely improve the transfer rate.

In a white-box scenario, other types of AEs may become applicable for generating evasive samples, such as Fast gradient method or Carlini&Wagner, for computing an AE in the features space and mapping it back to obtain an evasive document. However, most of the existing ML attacks were incapable of maintaining malicious signatures while easily evading classifiers, as shown in Table VII. Patterns that can easily be flipped by adversarial perturbations are known as nonrobust features [52]. The aforementioned AEs can easily compute adversarial perturbations to evade classifiers by flipping nonrobust features. However, none of these approaches use reconstruction loss to preserve to maintain original PDF behavior, which often resulted in a crash. Consequently, GAN reconstruction loss was necessary to conserve the original's PDF behavior.

The notion of robust and nonrobust features are defined by Ilyas *et al.* [52]. Robust features correspond to patterns that are predictive of the true label even when input is adversarially perturbed. Conversely, nonrobust features correspond to patterns that are also predictive but can easily be flipped by adversarial perturbations. ML models use both features to minimize the training loss; thus, flipping nonrobust features will have a huge impact on their prediction accuracy.

To mitigate AEs, Goodfellow *et al.* [16] introduced an algorithm, called adversarial training, that is robust to AEs by retraining them. In the same sense, antivirus vendors can prevent such adversarial attacks by collecting mutated examples and updating their detectors. However, once the detectors have been updated, attackers can also retrain PDF-GAN that exploit target detectors. It was observed that new AEs remained undetected by the updated detectors. In our experiments, among 500 mutation seed files, some evasive PDFs were successfully uploaded to the Gmail server. We believe that it is also possible to consider Gmail as the target detector under the strong black-box scenario.

In the process of denoting PDFs in a tree structure form and in the process of extracting and selecting a feature set from the tree structure, we observed that there is considerable loss of information concerning the PDFs. For example, the most recent detectors select few features from a large group of features to be used in the training phase. We believe that the performance of feature extraction on the PDFs is directly associated with the generation of performance. Therefore, eliminating the handcraft of such a process can increase the performance of the PDF detectors. A similar story is also true for the evasion scenario. Once all information can be represented and abstracted without the application of any handcraft for training GANs, a considerable increase in the performance of evading malware classifiers can be observed.

DL has been applied to several forms of high-dimensional data to denote a low-dimensional Euclidean space, and recently, DL has been expanded even to non-Euclidean spaces such as graphs [53]. Word2Vec is a representative algorithm that generates a low-dimensional embedded vector that corresponds to a high-dimensional word based on the mutual occurrence frequencies of words. Similarly, there are algorithms for embedding a graph [54] that maps a graph region to a low-dimensional region while preserving the adjacency and structure of the nodes.

Graph2Vec is one of the representative algorithms used for graph data-driven learning approaches. Embedding algorithms such as Word2Vec and Graph2Vec may be suitable candidates for denoting all features of the PDF without any handcraft.

In recent years, a few research has used a generative network to find AEs [55], [56]. However, none of these works verified that the generated evasive PDFs successfully retained the original malicious behavior. In our studies, we verified against traditional benchmark ML classifiers and the preservation of original malicious behavior testing against commercial AntiVirus engines.

VII. Conclusion

We introduced a novel approach for generating evasive PDF samples. In addition, we introduced a new technique for selecting a feature set that even includes unique features lurked in malicious files through pooling. As a consequence, we achieved a better detection performance than the state-of-the-art open-source PDF malware classifiers. Finally, our approach was based on the gray-box threat model, in which the attacker is extremely restricted with information concerning the target classifier. We evaluated our approach using over 10 000 PDF documents from VirusTotal and over 20 000 PDF documents from Contagio and successfully discovered evasive samples for all unique 500 selected samples. We analyzed our model on commercial virus engines in addition to the traditional benchmark. By generating evasive sample through PDF-GAN, we identified significant flaws in some AntiVirus engines in the black-box scenario attack. We determined that one antivirus engine may outperform the rest, but this does not guarantee that it would be the best detector. While evasion attacks are still possible on commercial AntiVirus engines, we suggest that the use of multiple detectors will prevent such attacks. Finally, we have focused only on PDF as a problem space in generating AEs. Therefore, we can expand this work, for a future study, to a different domains such as binary software as it is unclear if the same methodology can be applied.

References

[1] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. 10th Int Conf. Malicious Unwanted. Softw.*, 2015, pp. 11–20.

[2] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-Sec: Deep learning in android malware detection," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 371–372, 2014.

[3] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2009, vol. 9, pp. 8–11.

[4] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: Feature hashing malware for scalable triage and semantic analysis," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 309–320.

[5] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javaScript code," in *Proc. 19th Int Conf. World Wide Web*, 2010, pp. 281–290.

[6] M. A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, and N. Provos, "CAMP: Content-agnostic malware protection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2013.

[7] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna, "COMPA: Detecting compromised accounts on social networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2013.

[8] G. Stringhini, C. Kruegel, and G. Vigna, "Shady paths: Leveraging surfing crowds to detect malicious web pages," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 133–144.

[9] J. Schlumberger, C. Kruegel, and G. Vigna, "Jarhead analysis and detection of malicious Java applets," in *Proc. 28th Annu. Comput. Secur. Appl. Conf.*, 2012, pp. 249–257.

[10] P. Laskov and N. Šrndić, "Static detection of malicious JavaScript-bearing PDF documents," in *Proc. 27th Annu. Comput. Secur. Appl. Conf.*, 2011, pp. 373–382.

[11] G. Kakavelakis and J. Young, "Auto-learning of SMTP TCP transport-layer features for spam and abusive message detection," in *Proc. 25th Int. Conf. Large Installation Syst. Admin.*, 2011, p. 18.

[12] N. Šrndic and P. Laskov, "Detection of malicious PDF files based on hierarchical document structure," in *Proc. 28th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2013, pp. 1–16.

[13] C. Smutz and A. Stavrou, "Malicious PDF detection using metadata and structural features," in *Proc. 28th Annu. Comput. Secur. Appl. Conf.*, 2012, pp. 239–248.

[14] C. Smutz and A. Stavrou, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors," in *Proc. Netw. Distrib. Syst. Secur. Conf.*, 2016.

[15] N. Šrndić and P. Laskov, "Hidost: A static machine-learning-based detector of malicious files," *EURASIP J. Inf. Secur.*, vol. 2016, no. 1, 2016, Art. no. 45.

[16] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," vol. 1050, p. 20, 2015.

[17] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," 2016, *arXiv:1611.01236*.

[18] S. Alfeld, X. Zhu, and P. Barford, "Data poisoning attacks against autoregressive models," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 1452–1458.

[19] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," 2017, *arXiv:1702.02284*.

[20] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," 2017, *arXiv:1702.05983*.

[21] D. Maiorca, B. Biggio, and G. Giacinto, "Towards adversarial malware detection: Lessons learned from PDF-based attacks," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1–36, 2019.

[22] "SonicWall detects, reports dramatic rise in fraudulent PDF files in Q1 2019," 2019. [Online]. Available: https://www.sonicwall.com/news/sonicwall-detects-reports-dramatic- rise-in-fraudulent-pdf-files-in-q1-2019/

[23] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk, "Scriptless attacks: Stealing the pie without touching the sill," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 760–771.

[24] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 552–561.

[25] N. rndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *Proc. IEEE Symp. Secur. Privacy*, 2014, pp. 197–211.

[26] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 21–24.

[27] " M. Parkour, "16800 clean and 11960 malicious files for signature testing and research," 2013. [Online]. Available: http://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html

[28] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 119–133.

[29] " Mitre. cve-2018-9958," 2018. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-9958

[30] " Mitre. cve- 2013–2729," 2013. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2729

[31] " Mitre. cve-2010-3654," 2010. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3654

[32] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ML attacks in the problem space," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 1332–1349.

[33] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: From phenomena to black-box attacks using adversarial samples," 2016, *arXiv:1605.07277*.

[34] "Freedesktop.org. 2018. poppler," 2018. [Online]. Available: https://poppler.freedesktop.org/

[35] D. Maiorca, I. Corona, and G. Giacinto, "Looking at the bag is not enough to find the bomb: An evasion of structural methods for malicious PDF files detection," in *Proc. 8th ACM SIGSAC. Symp. Inf. Comput. Commun. Secur.*, 2013, pp. 119–130.

[36] Weilin xu, PDF-malware-parser: A fork of PDFRW aiming at parsing PDF malware," 2015. [Online]. Available: https://github.com/mzweilin/PDF-Malware-Parser

[37] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.
[38] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*. New York, NY, USA: Springer, 2012, pp. 9–48.
[39] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 807–814.
[40] " Mitre. cve-2008-2992," 2008. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2992
[41] " Mitre. cve-2010-0188," 2010. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0188
[42] " Mitre. cve-2010-2883," 2010. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2883
[43] " Mitre. cve-2011-2462," 2011. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2462
[44] "Mitre. cve- 2017–13056," 2017. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-13056
[45] "Exploit-database," 2020. [Online]. Available: https://www.exploit-db.com/
[46] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Anon. Int. Conf. Learn. Representations*, SanDego, 2015, *arXiv:1412.6980*.
[47] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2990–2999.
[48] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6467–6476.
[49] A. A. Rusu *et al.*, "Progressive neural networks," 2016, *arXiv:1606.04671*.
[50] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.
[51] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," 2016, *arXiv:1611.02770*.
[52] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 125–136.
[53] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. 2nd Int. Conf. Learn. Representations*, 2014, *arXiv:1312.6203*.
[54] G.E. Hinton, "Learning distributed representations of concepts," in *Proc. 8th Annu. Conf. Cogn. Sci. Soc.*, vol. 1, p. 12, Aug. 1986.
[55] Y. Li, Y. Wang, Y. Wang, L. Ke, and Y.-A. Tan, "A feature-vector generative adversarial network for evading PDF malware classifiers," *Inf. Sci.*, vol. 523, pp. 38–48, 2020.
[56] Y. Wang, Y. Li, Q. Zhang, J. Hu, and X. Kuang, "Evading PDF malware classifiers with generative adversarial network," in *Proc. Int. Symp. Cyberspace Safety Secur.*, 2019, pp. 374–387.

Ho Bae received the B.Sc. degree in computer science and the M.Sc. degree in information security from University College London, London, U.K., in 2007 and 2009, respectively, and the Ph.D. degree in bioinformatics from Seoul National University, Seoul, South Korea, in 2021.

He was with SAP Labs Korea, Inc., Seoul, from 2012 to 2016. He is currently an Assistant Professor with the Department of Cyber Security, Ewha Womans University, Seoul. His research interests include artificial-intelligence-based security, bioinformatics, and privacy.

Younghan Lee received the B.Eng. degree in electrical and electronic engineering from Imperial College London, London, U.K., in 2016. He is currently working toward the Ph.D. degree in electrical and computer engineering with Seoul National University, Seoul, South Korea.

His research interests include robust artificial intelligence and adversarial examples in deep learning models.

Yohan Kim received the B.S. degree in computer science and engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 2019, and the M.Sc. degree in electrical and computer engineering from Seoul National University, Seoul, South Korea, in 2021.

His research interests include practical machine learning, deep learning, and anomaly detection.

Uiwon Hwang received the B.S. degree in biomedical engineering from Korea University, Seoul, South Korea, in 2016. He is currently working toward the Ph.D. degree in electrical and computer engineering with Seoul National University, Seoul.

His research interests include deep generative models, biomedical data science, secure artificial intelligence, and machine learning.

Sungroh Yoon (Senior Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 2002 and 2006, respectively.

From 2016 to 2017, he was a Visiting Scholar with the Department of Neurology and Neurological Sciences, Stanford University. He held research positions with Stanford University and Synopsys, Inc., Mountain View, CA. From 2006 to 2007, he was with Intel Corporation, Santa Clara, CA. He was an Assistant Professor with the School of Electrical Engineering, Korea University, Seoul, from 2007 to 2012. He is currently a Professor with the Department of Electrical and Computer Engineering, Seoul National University. His current research interests include machine learning and artificial intelligence.

Dr. Yoon was the recipient of the Seoul National University Education Award in 2018, the IBM Faculty Award in 2018, the Korean Government Researcher of the Month Award in 2018, the BRIC Best Research of the Year in 2018, the IMIA Best Paper Award in 2017, the Microsoft Collaborative Research Grant in 2017, the SBS Foundation Award in 2016, the IEEE Young IT Engineer Award in 2013, and many other prestigious awards.

Yunheung Paek (Member, IEEE) received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, South Korea, in 1988 and 1990, respectively, and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1997.

He is currently a Professor with the Department of Electrical and Computer Engineering, Seoul National University. His research interests include system security with hardware, secure processor design against various types of threats, and machine-learning-based security solution.