IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2017.Doi Number*

# MalJPEG: Machine Learning Based Solution for the Detection of Malicious JPEG Images

**Aviad Cohen[a], Nir Nissim[a,b], and Yuval Elovici[b]**

[a] Malware Lab, Cyber Security Research Center, Ben-Gurion University of the Negev, Israel
[b] Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Israel
[c] Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Israel

Corresponding author: Aviad Cohen (e-mail: aviadjo@gmail.com).

**Abstract** In recent years, cyber-attacks against individuals, businesses, and organizations have increased. Cyber criminals are always looking for effective vectors to deliver malware to victims in order to launch an attack. Images are used on a daily basis by millions of people around the world, and most users consider images to be safe for use; however, some types of images can contain a malicious payload and perform harmful actions. JPEG is the most popular image format, primarily due to its lossy compression. It is used by almost everyone, from individuals to large organizations, and can be found on almost every device (on digital cameras and smartphones, websites, social media, etc.). Because of their harmless reputation, massive use, and high potential for misuse, JPEG images are used by cyber criminals as an attack vector. While machine learning methods have been shown to be effective at detecting known and unknown malware in various domains, to the best of our knowledge, machine learning methods have not been used particularly for the detection of malicious JPEG images. In this paper, we present MalJPEG, the first machine learning-based solution tailored specifically at the efficient detection of unknown malicious JPEG images. MalJPEG statically extracts 10 simple yet discriminative features from the JPEG file structure and leverages them with a machine learning classifier, in order to discriminate between benign and malicious JPEG images. We evaluated MalJPEG extensively on a real-world representative collection of 156,818 images which contains 155,013 (98.85%) benign and 1,805 (1.15%) malicious images. The results show that MalJPEG, when used with the LightGBM classifier, demonstrates the highest detection capabilities, with an area under the receiver operating characteristic curve (AUC) of 0.997, true positive rate (TPR) of 0.951, and a very low false positive rate (FPR) of 0.004.

**INDEX TERMS** JPEG, Image, Malware, Detection, Machine Learning, Features.

## I. INTRODUCTION

Cyber attacks targeting individuals, businesses, and organizations have increased in recent years. *Infosecurity* magazine declared that cyber attacks doubled in 2017.[1] Cyber attacks usually include harmful activities such as stealing confidential information, spying, or monitoring, and cause harm (sometimes significant) to the victim. Attackers may be motivated by ideology, criminal intent, a desire for publicity, etc.

Attackers are constantly searching for new and effective ways to launch attacks and deliver a malicious payload to victims. Files sent via the Internet have often served as a means of accomplishing this. Since executable files (i.e., *.exe) are known to be dangerous, attackers are increasingly using non-executable files (e.g., *.pdf, *.docx, etc.) which are mistakenly considered to be safe for use by most users. Some non-executable files allow an attacker to run arbitrary malicious code on the targeted victim machine when the file is opened.

JPEG (Joint Photographic Experts Group) is the most popular image format [2], mainly because of its lossy compression. JPEG images are used by almost everyone, from individuals to large enterprises, and on various platforms. JPEG images can be found on computers (personal images, documents), devices (smartphones, digital cameras, etc.), and in cyber space (emails, social media, websites, etc.). Due to their harmless reputation, massive use, and high potential for misuse, cyber criminals use JPEG images as an attack vector in order to deliver their malicious payload to the victim device.

At the 2015 Black Hat conference, Saumil Shah demonstrated[3] how to create malicious JPEG images that can be loaded in a browser in order to execute malicious code automatically.[4,5] In November 2016, it was reported that attackers used *Facebook* Messenger to spread the infamous *Locky* ransomware via JPEG images.[6] The malware authors discovered security vulnerabilities in *Facebook* and *LinkedIn* that allow them to forcibly download a malicious image on the victim's computer. In August 2017, it was reported that *SyncCrypt* ransomware was spread using JPEG images.[7] In December 2018, Trend Micro,[8] an enterprise cyber security company, reported that cyber criminals used memes on *Twitter* (JPEG images) in order to convey commands to malware.[9] Recently, in December 2019, researchers from the Sophos security company published a comprehensive report[10] on the *MyKings* cryptomining botnet that lurks behind a seemingly innocuous JPEG of Taylor Swift.

The ability to detect malicious JPEG images has great importance as JPEG images are widely used by individuals and businesses. Existing endpoint defense solutions which are based on signatures (e.g., antivirus), can only detect known malware based on their signature database. When a new malware, or new variant of existing malware appears, there is a time lag until these defense solutions update their clients with the new signature—a time in which the clients are vulnerable to the new malware. In contrast, in recent years, machine learning (ML) algorithms have demonstrated their ability to detect both known and unknown malware in various domains, particularly for the detection of malware in various types of files [1]–[7]. However, to the best of our knowledge, machine learning methods have not been employed for the detection of malicious JPEG images.

In this paper, we present *MalJPEG*, a machine learning-based solution for efficient detection of unknown malicious JPEG images. *MalJPEG* extracts 10 simple but discriminative features from the JPEG file structure and leverages them with a machine learning classifier, in order to discriminate between benign and malicious JPEG images. We evaluate *MalJPEG* extensively on a real-world representative collection of benign and malicious JPEG images. We also compare *MalJPEG* features to features extracted using several existing generic feature extraction methods.

The paper's contributions are as follows:

1. *MalJPEG* – a machine learning based solution for efficient detection of known and unknown malicious JPEG images.
2. *MalJPEG* features – a compact set of 10 simple yet discriminative features for the efficient detection of malicious JPEG images using machine learning techniques.
3. The creation of a large and representative labeled collection of benign and malicious JPEG images that can be used for further research by the scientific community.

We provide background information related to the JPEG file format in Section II and discuss related work in Section III. Section IV describes the methods used in this research and the *MalJPEG* features. We evaluate our method and present the results in Section V. We discuss the results and various aspects of security and present our conclusions in Section VI.

[1] https://www.infosecurity-magazine.com/news/cyberattacks-doubled-in-2017/

[2] https://1stwebdesigner.com/image-file-types/

[3] https://www.blackhat.com/docs/eu-15/materials/eu-15-Shah-Stegosploit-Exploit-Delivery-With-Steganography-And-Polyglots.pdf

[4] https://www.opswat.com/blog/hacking-pictures-stegosploit-and-how-stop-it

[5] https://www.opswat.com/blog/image-borne-malware-how-viewing-image-can-infect-device

[6] https://thehackernews.com/2016/11/facebook-locky-ransomware.html

[7] https://www.bleepingcomputer.com/news/security/synccrypt-ransomware-hides-inside-jpg-files-appends-kk-extension/

[8] https://www.trendmicro.com

[9] https://blog.trendmicro.com/trendlabs-security-intelligence/cybercriminals-use-malicious-memes-that-communicate-with-malware/

[10] https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophoslabs-uncut-mykings-report.pdf

## II. Background

In this section, we provide background material related to our research, as well as technical information regarding the structure of a JPEG image. Since the JPEG file structure is complicated, we only present the basic information needed to enable the reader to comprehend the paper and understand the proposed *MalJPEG* solution presented in this research. The format of JPEG images is comprehensively described in the JPEG File Interchange Format (JFIF) specification.[11]

### A. JPEG File Structure

JPEG stands for Joint Photographic Experts Group, which has become the most popular image format on the Web. In 1992, JPEG became an international standard for compressing digital still images. JPEG files usually have a filename extension of *.jpg or *.jpeg.

A JPEG image file is a binary file which consists of a sequence of segments. Segments can be contained in other segments hierarchically. Each segment begins with a two-byte indicator called a "marker." The markers help divide the file into different segments. A marker's first byte is *0xFF* (hexadecimal representation); the second byte may have any value except *0x00* and *0xFF*. The marker indicates the type of data stored in the segment. Segment types are assigned names based on their definition or purpose; for example, the name of *0xFFD9* is *EOI*, and the name of *0xFFFE* is *COM*. Segment types *0xFF01* and *0xFFD0-0xFFD9* consist entirely of the two-byte marker; all other markers are followed by a two-byte integer indicating the size of the segment, followed by the payload data contained in the segment. Table 1 presents the possible markers, their hexadecimal code, and their definition/purpose.

A JPEG image begins with the *0xFFD8* maker (*SOI* – start of image) which is followed immediately by the *0xFFE0* marker ($APP_0$). A JPEG image ends with *0xFFD9* (*EOI* – end of image). Figure 1 presents the hexadecimal view of a sample JPEG image file; the bold bytes are the markers.

JPEG image files primarily use two classes of segments: marker segments and entropy-coded segments. **Marker segments** contain general information (metadata) such as header information and tables (quantization tables, entropy-coding tables, etc.) required to interpret and decode the compressed image data. **Entropy-coded segments** contain the entropy-coded data (follows the *SOS* marker). The compressed content inside a JPEG image is placed inside a sequence of units called a *frame*. A *frame* is a collection of one or more *scan* units. A *scan* contains a complete encoding of one or more image components.[12]

Figure 2 presents the structure of a simple JPEG image file and the hierarchy of the markers and their division into frames and scans. The markers in bold are mandatory or the most common markers.

| Marker Name | Hexadecimal Code | Definition/Purpose |
|---|---|---|
| **$APP_n$** | 0xFFE0-0xFFEF | Reserved for **app**lication used |
| **COM** | 0xFFFE | **Com**ment |
| **DAC** | 0xFFCC | **D**efine **a**rithmetic **c**onditioning table(s) |
| **DHP** | 0xFFDE | **D**efine **h**ierarchical **p**rogression |
| **DHT** | 0xFFC4 | **D**efine **H**uffman **t**able(s) |
| **DNL** | 0xFFDC | **D**efine **n**umber of **l**ines |
| **DQT** | 0xFFDB | **D**efine **q**uantization **t**able(s) |
| **DRI** | 0xFFDD | **D**efine **r**estart **i**nterval |
| **EXP** | 0xFFDF | **Exp**and reference image(s) |
| **JPG** | 0xFFC8 | Reserved for **JPE**G extensions |
| **$JPG_n$** | 0xFFF0-0xFFFD | Reserved for **JPE**G extensions |
| **RES** | 0xFF02-0xFFBF | **Res**erved |
| **$RST_m$** | 0xFFD0-0xFFD7 | **Rest**art with modulo 8 counter m |
| **$SOF_n$** | 0xFFC0-3, 5-7, 9-B, D-F | **S**tart **o**f ***Frame*** |
| **SOS** | 0xFFDA | **S**tart **o**f ***Scan*** |
| **TEM** | 0xFF01 | For **tem**porary use in arithmetic coding |
| **SOI** | 0xFFD8 | **S**tart **o**f **i**mage |
| **EOI** | 0xFFD9 | **E**nd **o**f **i**mage |

**Table 1. Possible JPEG markers.**

```
00000000 FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48
00000010 00 48 00 00 FF DB 00 43 00 04 03 03 03 03 02 04
00000020 03 03 03 04 04 04 05 06 0A 06 06 05 05 06 0C 08
00000030 09 07 0A 0E 0C 0F 0E 0E 0C 0D 0D 0F 11 16 13 0F
00000040 10 15 11 0D 0D 13 1A 13 15 17 18 19 19 19 0F 12
00000050 1B 1B 1B 18 1D 16 18 19 18 FF DB 00 43 01 04 04
00000060 04 06 05 06 0B 06 06 0B 18 10 0D 10 18 18 18 18
00000070 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
00000080 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
00000090 18 18 18 18 18 18 18 18 18 18 18 18 18 FF C0
000000A0 00 11 08 01 F4 01 E8 03 01 22 00 02 11 01 03 11
000000B0 01 FF C4 00 1D 00 00 01 04 03 01 01 00 00 00 00
000000C0 00 00 00 00 00 00 04 02 03 05 06 00 07 08 01 09
000000D0 FF C4 00 40 10 00 02 01 03 03 02 05 03 03 02 05
000000E0 02 05 03 05 01 01 02 03 00 04 11 05 12 21 06 31
000000F0 07 13 22 41 51 08 61 71 14 32 81 42 91 15 23 52
00000100 A1 B1 33 62 16 24 C1 D1 E1 17 72 F1 09 25 43 53
00000110 F0 82 FF C4 00 1C 01 00 02 02 03 01 01 00 00 00
00000120 00 00 00 00 00 00 00 03 04 02 05 01 06 07 00 08
00000130 FF C4 00 39 11 00 02 02 02 01 03 03 02 05 02 05
   ...                          ...
0000B600 FF D9
```

**Figure 1. JPEG file structure in hexadecimal view; the bold bytes are markers.**

[11] https://www.w3.org/Graphics/JPEG/jfif.txt

[12] https://www.w3.org/Graphics/JPEG/itu-t81.pdf

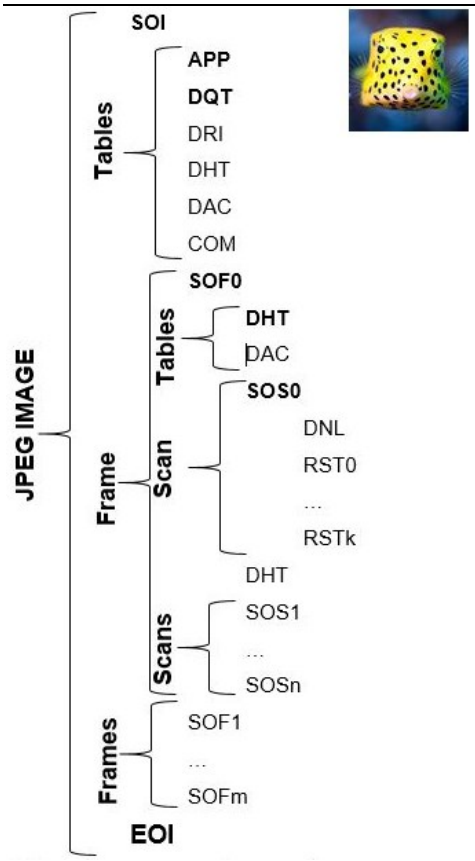


**Figure 2. The structure of a simple JPEG image and the hierarchy of the markers and their division into frames and scans. The markers in bold are mandatory or the most common markers.**

### *B. Embedding Malicious Payload in JPEG Images*

**Vulnerability Exploitation** – No software is ever completely protected, and it is almost impossible to prevent the presence of vulnerabilities during the development of a large-scale software project. Such vulnerabilities, when exploited, can allow an adversary to obtain higher privileges or divert the normal execution flow to an arbitrary malicious code. In addition, in order to view/parse a JPEG image, a viewer/parser program is required, and these programs may have some vulnerabilities. Many vulnerabilities related to JPEG images have been discovered since it was first published, and there are currently 303 known vulnerabilities [13] (CVE – Common Vulnerabilities and Exposures), and 5,520 known related security issues[14] associated with JPEG images. For example, a recently discovered vulnerability (CVE-2018-6612) may allow a remote attacker to cause a denial-of-service when the victim processes a malicious JPEG file.[15]

**Steganography** (steganos – covered, graphie – writing) – Steganography,[16] a technique used for disguising information (e.g., text or malicious code) inside the image without affecting its appearance (invisible to the human eye) is very difficult to detect. Steganography can be used to exfiltrate sensitive information from the victim's host or network via JPEG images and can even be used for delivering pieces of code into the victim's host or network under the guise of a simple benign JPEG image.

It is important to emphasize that malicious JPEG images do not necessarily use steganography methods to conceal the embedded payload; thus, we discriminate between JPEG images that carry hidden information using steganography and JPEG images that carry a malicious payload. In this research, we focus only on the later and the detection of malicious JPEG images, and not on steganography detection.

## III. Related Work

The main domain of this research is malware detection using machine learning. Many studies have already been performed on malware detection in various operating systems and file types for both executable and non-executable files. For example, [6], [8]–[13] proposed solutions for the detection of malware in executable files using machine learning. In 2019, [14] presented a survey of machine learning techniques for malware analysis in Windows portable executables (PEs). They reviewed previous work considering the objective of the analysis, the feature extraction technique used (static, dynamic, or hybrid analysis), the type of features extracted from samples, and the machine learning models used. In 2019, [15] presented a comprehensive, state-of-the-art survey of existing methods for the dynamic malware analysis of PE files, including machine learning-based methods. With regard to non-executable files, [4], [7], [16]–[19] proposed ML-based solutions for the detection of malicious PDF documents; [1], [2], [20] proposed ML-based solutions for the detection of malicious Office documents; [21] proposed an ML-based solution for the detection of malicious SWF (Flash) files; and [3] proposed an ML-based solution for the detection of malicious emails.

In the course of this research we sought out related work aimed at the detection of malicious images, and JPEG images in particular, using machine learning. However, we were unable to find any papers that apply machine learning for the detection of malicious JPEG images. It is important to emphasize that we only refer to images that contain malware or malicious code as *malicious images*. Therefore, the domain of adversarial image detection (e.g., [22]–[26]) is different than this paper's domain. The difference between adversarial images and malicious images lies in the part of the images that is altered to transform an original image into an adversarial or malicious image. Adversarial images are created by intentionally designing the pixels of the original image in such a way that the image will be misclassified by a machine learning model. In contrast, malicious JPEG images store the malicious mechanism in some metadata fields which are outside the pixels section; usually in

[13] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=JPEG
[14] https://www.cvedetails.com/google-search-results.php?q=JPEG&sa=Search
[15] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-6612
[16] https://searchsecurity.techtarget.com/definition/steganography

this case, the pixels are not changed in order to maintain the image's authenticity.

Kunwar et al. [27], proposed a theoretical framework that is aimed to detect the presence of data or code in JPEG images (without the use of machine learning). Their framework has three phases: 1) steganography analysis, 2) extraction of the embedded file, and 3) uploading the extracted file to an online scanning tool such as *VirusTotal*[17] or *Metascan*.[18] The paper does not, however, describe any experiments performed in order to evaluate the framework or present any detection results on a real-world dataset. In addition, we identified a study [28], which proposed an authentication method for JPEG images that can distinguish legitimate operations (e.g., compression) from malicious operations. However, "malicious" in the context of this paper does not mean that that image carries malicious code as a payload, but rather that the image is not authentic and that it has been manipulated. In addition, the paper does not apply machine learning methods.

Most of the papers we found on JPEG images focused on steganography methods [29]–[32], steganography analysis (steganalysis) methods [33]–[37], or adversarial images [38]–[42]. However, this paper focuses on the detection of malicious JPEG images. To the best of our knowledge, machine learning methods have not been used for the detection of malicious JPEG images, and thus we are the first to propose a machine learning-based solution tailored specifically for the detection of malicious JPEG images.

## IV. Methods

In this section, we describe the methods used in this research. We begin by presenting *MalJPEG*'s features as well as the existing generic feature extraction methods. We then compare the features extracted by the existing generic feature extraction methods and the features extracted by the *MalJPEG* feature extractor. Finally, we describe the machine learning algorithms we used in this research.

### A. MalJPEG Solution

In this section, we present the core contribution of our paper, the *MalJPEG* machine learning-based solution for the detection of malicious JPEG images. *MalJPEG* receives a JPEG image as input {1}. The *MalJPEG* feature extractor {2} extracts the proposed features into a vector of features {3}. The *MalJPEG* feature extractor inspects the file statically, without actually viewing the image (which requires executing image viewer software that itself might have a vulnerability), and traverses through the JPEG image file structure in order to extract the features. The features are then transferred to a pretrained machine learning-based model {4} which outputs a classification (benign/malicious) {5} for the input image. We implemented *MalJPEG* and its inner modules, the feature extractor {3} and machine learning model, {4} in Java programming language. The next section provides a detailed description of the features that are extracted using *MalJPEG*.

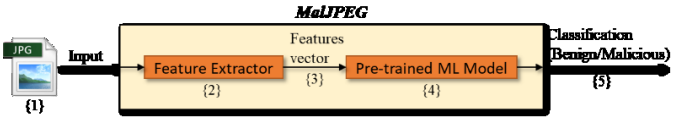


**Figure 3. The *MalJPEG* Solution**

#### 1) MalJPEG Features

In this section, we present the compact set of discriminative features extracted by *MalJPEG*. We engineered these features after manually examining the structure of many benign and malicious JPEG images. We gained an understanding of how attackers use JPEG images in order to launch attacks and how it affects the JPEG file structure. We also found how malicious JPEG images differ from regular benign JPEG images in terms of file structure. For example, some malicious JPEG files contain data (usually code) after the end-of-file (*EOI*) marker. In addition, we statistically analyzed the distribution for JPEG markers' frequency and size in both malicious and benign JPEG images and define features that primarily discriminate between benign and malicious JPEG images.

The features are very simple, and most of them are based on the presence and size of specific markers within the JPEG image file structure. In addition, the features are relatively easy to extract statically (without actually presenting the image) when parsing the JPEG image file. Table 2 contains the set of *MalJPEG* features; the features are sorted by their Information Gain [43] rank so that the first feature is the most prominent. Note that, all of the features are numeric.

| # | Feature Name | Description | Info Gain Rank |
|---|---|---|---|
| 1 | Marker_EOI_content after_num | Number of bytes after the EOI (end of file) marker. | 0.058 |
| 2 | Marker_DHT_size_max | Maximal DHT marker size found in the file. | 0.025 |
| 3 | File_size | Image file size in bytes. | 0.023 |
| 4 | Marker_APP1_size_max | Maximal APP1 marker size found in the file. | 0.023 |
| 5 | Marker_COM_size_max | Maximal COM marker size found in the file. | 0.017 |
| 6 | Marker_DHT_num | Number of DHT markers found in the file. | 0.016 |
| 7 | File_markers_num | Total number of markers found in the file. | 0.014 |
| 8 | Marker_DQT_num | Number of DQT markers found in the file. | 0.012 |
| 9 | Marker_DQT_size_max | Maximal DQT marker size found in the file. | 0.012 |
| 10 | Marker_APP12_size_max | Maximal APP12 marker size found in the file. | 0.011 |

**Table 2. MalJPEG features, sorted by their prominence according to their Information Gain rank.**

Information Gain ranks a feature (attribute) by measuring the reduction in entropy of a given set after dividing it based on a specific feature; this results in the gain of information as a result of using the feature. Specifically, it subtracts the weighted entropies of each subset from the original entropy of

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

[18] https://www.opswat.com/blog/tag/metascan-online

the entire set. The entropy characterizes the disorder in an arbitrary set of instances. If the set is completely homogeneous, the entropy is zero; if the set is equally divided, then it has entropy of one. Information Gain assigns a higher rank to features that contribute significantly to discrimination between malicious and benign classes. Equation 2 presents the formula of the entropy of a set of items $S$ divided into $C$ subsets (classes). $p_i$ represents the probability of class $C$ in a set of items. Equation 1 presents the Information Gain formula for feature $F$ in a set of items $S$ where $V$ is the set of possible values of $A$.

$$IG(S,F) = E(S) - \sum_{v \epsilon V(F)} \frac{|S_v|}{|S|} \cdot E(S_v) \qquad E(S) = \sum_{c \epsilon C} -p_i \cdot \log_2(p_i)$$

**Equation 1** **Equation 2**

A malicious JPEG image carries the malicious payload within itself in some way. Therefore, some of the proposed features are indicative of the maximal size of specific markers (e.g., DHT, DQT, COM, APP1, APP12) that are used by attackers to store the malicious payload. The injection of a payload into a marker increases its size beyond the typical size. In some cases, the malicious payload is spread across multiple markers. Therefore, the reset of the features are indicative of the frequency of specific markers in the file.

An attempt to make a benign file malicious will affect the structure of the image file and thus will be reflected in the selected features. It is important to mention that it is possible that future attack techniques applied on malicious JPEG images will use markers that are not covered by *MalJPEG* features. Such attacks will likely still affect the structure of the image file and therefore be reflected in the existing selected features which cover the most important markers. In any case, the *MalJPEG* feature extractor is extendable and can easily be updated to extract new features representative of any other marker in the image file that may be found to be important in the future.

Figure 4 presents a histogram of the values of the *'Marker_DQT_num'* feature (numeric) among the benign and malicious classes. The *x*-axis represents the number of DQT markers (bin size = 10) in an image, and the *y*-axis represents the percentage of images from the class (benign or malicious) that fall in that bin. It can be seen that the distribution is completely different for benign and malicious classes, and this is most noticeable when looking at the range of the values. In addition, 99.21% of the benign images have between zero and nine DQT markers, whereas only 13.74% of the malicious images have between zero and nine DQT markers, and 38.49% of them have between 10,200 and 10,209 DQT markers. In Figure 4 we can see that the DQT marker is much more prevalent in malicious images than in benign images. The DQT marker is, in fact, used by attackers to store the malicious payload. Therefore, the *'Marker_DQT_num'* feature is prominent and assists in differentiating between benign and malicious images.

Figure 5 presents a heat map of the Pearson correlation matrix of *MalJPEG* features; low correlation is cooler, and high correlation is warmer. Each cell inside the matrix represents the correlation between two features; a correlation is a number between zero and one. One can see that there are no redundant features and that in general, the features are not highly correlated with each other. Although the features (*Marker_DHT_num*, *File_marker_num*, and *Marker_DQT_num)* have correlation scores between 0.76 and 0.78, each of them still hold additional valuable information that benefits the machine learning classifier; thus, they are all necessary. The average correlation between the features (the lower triangle under the diagonal) is 0.267.

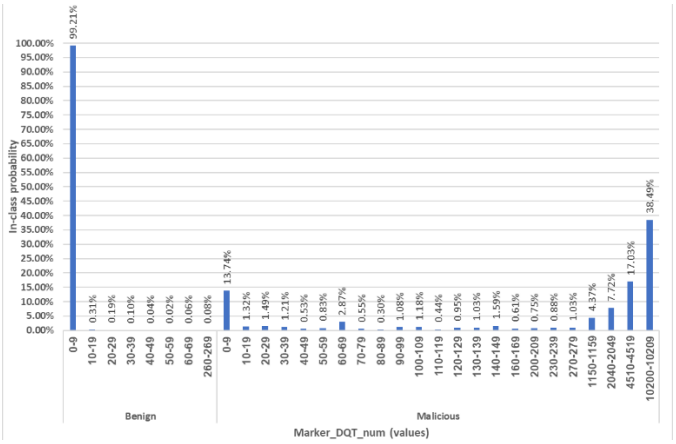


**Figure 4. A histogram of the DQT marker values among benign and malicious images.**

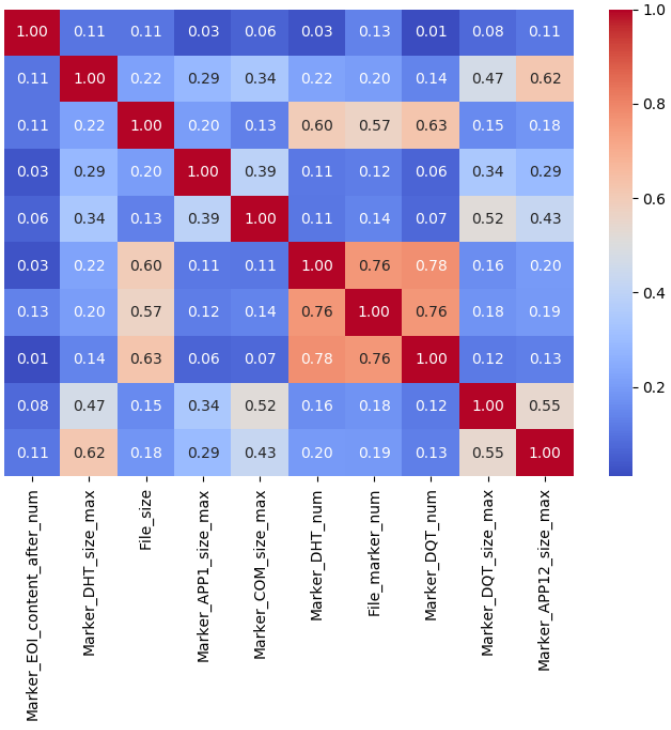


**Figure 5. Heat map of the Pearson correlation matrix of the MalJPEG features.**

### *B. Generic Feature Extraction Methods*

In this section, we present generic and static feature extraction methods that were used in previous academic work

in conjunction with machine learning for malware detection. The advantage of generic feature extraction methods is that they model the contents of a file in a file-format agnostic way. Generic feature extraction methods can be applied on any file format, in contrast to the *MalJPEG* feature extractor which is tailored to JPEG images. Generic feature extraction methods work on the file's building blocks (byte or character representation) in order to extract features that represent the file. It is important to clarify that we did not apply the generic methods on pixel values (although it is possible to do so), because the pixels only hold the image's visual properties of the file but lack the file's metadata. In general, malicious content that is injected into an image by an attacker is stored in the file's metadata; thus, it is important to inspect the JPEG file as a whole, and not only part of it.

*1) Histogram*

A histogram feature extractor creates a fixed-size histogram, built according to the file's content; the histogram values can be used as features for machine learning algorithms. In this research, we use two types of histograms: a **simple histogram** and an advanced **byte entropy histogram**.

Two **simple histogram** configurations are used: 1) a histogram on byte values (256 options), and 2) a histogram on character values (128 options, for basic ASCII encoding). The histogram actually counts the frequency of byte or char values in the file. In order to be able to compare files of different sizes, we normalize the histogram values between zero and one.

We also use the **byte entropy histogram** technique which was used for malware detection in prior research [44], [45]. In order to extract the byte entropy histogram, we slide a *K*-size byte window, with a stride of *S* bytes, over the byte array representation of a file. For each window, we compute the base 2 entropy of the window. We store each individual byte value in the window (*K* non-unique values) with the entropy value of the whole window (as a pair) in a list; in total, there are *K* pairs for each window. Finally, we compute a two-dimensional histogram over the pair list, where the histogram entropy axis has *E* evenly sized bins over the range [0, *E*], and the byte axis has *B* evenly sized bins over the range [0, *B*]. To obtain a feature vector (rather than the matrix), we concatenate each row vector in this histogram into a single *E*B* sized feature vector. Usually, *B* is set to 256 (the number of unique byte values), and *E* is set to 8, 16, or 256. The intuition behind the byte entropy histogram approach is that it represents byte values in the entropy context.

*2) Min-Hash*

Min-Hash [46] is a technique for quickly estimating how similar two items are. The Min-Hash method generates an *N*-size signature for a given file based on *N* simple hash functions (fuzzy hashing). The similarity of two items can be easily computed by calculating the *Hamming* distance between their signatures; the more matched signatures, the lower the *Hamming* distance. The signature created by the Min-Hash technique can be used as an input feature for lazy machine learning algorithms which are based on a distance function (e.g., *K-Nearest Neighbors*).

Min-Hash signs a file (generates a signature) based on *shingles* extracted from the file. A shingle is a fixed length sequence of basic units in the file (e.g., byte, char, etc.). A basic way of extracting shingles is to slide a *W*-size window, with a stride of *S*, over the file. *N* hash functions are applied on each shingle extracted from a file, and the hash results (Long type) are stored on an *N*-size array. The signature of the file is actually an *N*-size vector of Long numbers, which contains only the minimal hash value produced by each hash function, across all shingles. Min-Hash is initialized with the following parameters: the shingle size *W*, stride *S*, and number of hash functions *N*.

The Min-Hash method is very efficient in terms of time and space. Any file, of any size, can be easily converted to an *N*-size signature. The Min-Hash method has previously been used for malware detection [47]–[50] and malware clustering [51].

### C. Machine Learning Algorithms

We applied machine learning classification algorithms on the datasets described in the previous section. In our experiments, we utilized the following commonly used, high performing classic and nonlinear machine learning classifiers: *Decision Tree*, *Random Forest,* and *Gradient Boosting on Decision Trees* (*XGBoost* and *LightGBM*). We chose these classifiers as they perform well on highly imbalanced datasets. In is important to mention that in our preliminary experiments we examined classifiers from families other than the decision tree family, such as Logistic Regression and Naïve Bayes, however they did not provide reasonable results; therefore, we did not include them in our evaluation.

In addition, we used the *K-Nearest Neighbors* classifier (K=5) on Min-Hash datasets, because it is the only classifier that can actually compare Min-Hash signatures using the *Hamming* distance function. We chose to u

We applied the abovementioned machine learning classifiers with Python using the following packages: *scikit-learn,*[19] *XGBoost,*[20] and *LightGBM.*[21] We used the default configuration for all classifiers.

## V. Evaluation

In this section, we evaluate *MalJPEG*. We begin by presenting our data collection which we used for evaluation, and then describe the dataset creation process. Then, we present our research questions, evaluation metrics, experimental design, and results.

### A. Data Collection

We obtained a large collection of unique benign and malicious JPEG images. The benign images were collected from social media (*Facebook*, *Instagram*, *WhatsApp*, etc.); we focus on viral images of different file sizes and on different topics (memes, food, personal photos from social media, etc.). We

[19] http://scikit-learn.org/stable/

[20] https://xgboost.readthedocs.io/en/latest/

[21] https://lightgbm.readthedocs.io/en/latest/

verified that the images are benign by scanning them using *VirusTotal*. We make the assumption that these images do not contain unknown threats. The malicious images were collected from *VirusTotal.* We only used JPEG images that were labeled as malicious by at least five (out of 69) antivirus engines. We set our threshold at five, a level that we feel is sufficient for determining whether a file is malicious for the following reasons. 1) Antivirus engines primarily rely on known signatures of malware and have zero false positives; if a malware signature is found in a file, the file almost certainly contains malware. When combining five different reliable antivirus engines, the certainty increases. 2) A threshold of five was also used in previous research in the field of malware detection [52]–[54]. The query we used to retrieve malicious JPEG images from *VirusTotal* is: *'p:5+ and (type:jpeg) fs:2016-01-01+'*. This query searches *VirusTotal* for JPEG files uploaded since the beginning of 2016 that were labeled as malicious by at least five antivirus engines. Additional information *on VirusTotal* search modifiers can be found on their website. [22]

We verified that all of the files in our collection were actual images using an automated code that we wrote which verifies that a file has JPEG characteristics. Our collection includes 156,818 images in total: 155,013 (98.9%) benign and 1,805 (1.15%) malicious, which were collected between 2016 and 2018.

Note that the percentage of malicious images in our collection is extremely low (1.15%). We prepared our collection that way intentionally, so that the collection reflects (as much as possible) the ratio between malicious and benign JPEG images in the real world. According to Moskovitch [55] who dealt with the issue of imbalanced collections in the cyber security domain, the most realistic detection performance is achieved when the percentage of malicious files in the training set is equal to the percentage in the real world. Note also that the extremely low percentage of malicious instances (positive) in the collection makes the detection of malicious images extremely difficult.

*Figure 6* presents the distribution of threats in our malicious JPEG image collection, in a pie chart, sorted from the most common threat included in the collection to the most uncommon threat. We derived the specific type of threat for each malicious file by inspecting the identification provided by different antivirus engines from the *VirusTotal* report for each file. As can be seen, our malicious JPEG dataset contain a diverse range of threats. 36% of the images use an Iframe HTML tag for malicious purposes, 26% contain malicious PHP code, 10% contain a Trojan, 6% use JavaScript code, 6% exploit some vulnerability in the system, 5% contain a virus, 4% contain HTML code for malicious purposes, 4% contain some malicious script, 2% contain a cryptocurrency miner, and 1% contain ransomware. One percent of the malicious files target the Android OS.

### B. Dataset Creation

We wrote a Java program which creates a dataset (i.e., a dataset creator program). The input for the program is a Java implementation of a feature extraction method. The dataset creator applies the feature extractor (with the desired configuration) on all of the files in a given collection and combines the feature vectors into a final *.CSV dataset. We implemented all of the abovementioned generic feature extraction methods, as well as a feature extractor which extracts the *MalJPEG* features.

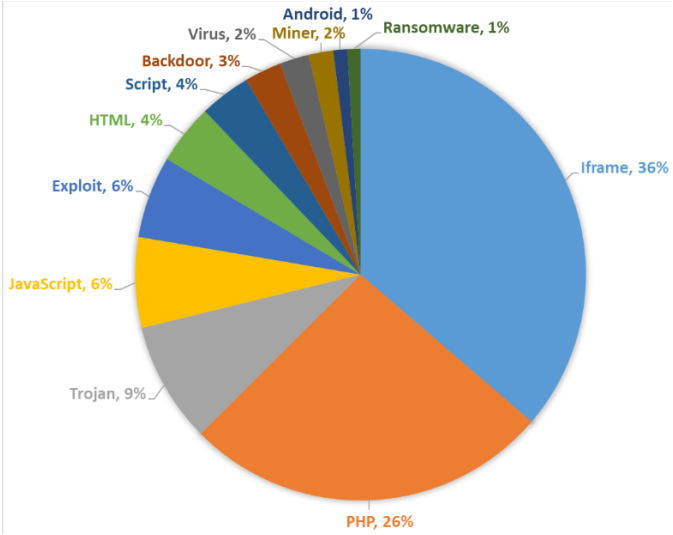


**Figure 6. Distribution of threats in our malicious collection.**

In order to compare the *MalJPEG* features with features extracted using generic feature extraction methods, we created various datasets. Each dataset represents a specific feature extraction method with a specific configuration. To accomplish this, we applied our dataset creator program with all of the abovementioned feature extractors and various configurations, on our JPEG image collection, in order to create a collection of datasets.

### C. Research Questions

In order to evaluate the effectiveness of *MalJPEG* in detecting malicious JPEG images, we designed experiments to answer the following research questions:

1. Can machine learning-based classifiers that have been trained on *MalJPEG* features efficiently detect unknown malicious JPEG images?
2. Do machine learning classifiers that have been trained on *MalJPEG* features provide better detection results than machine learning classifiers that have been trained on features extracted using histogram and Min-Hash generic feature extraction methods?
3. How does the *MalJPEG* feature extractor perform in comparison to the generic feature extraction methods in terms of time complexity?
4. Which classifier provides the best detection results with *MalJPEG* features: *Decision Tree*, *Random Forest*, *LightGBM, XGBoost* or *K-Nearest Neighbors?*
5. How does the *MalJPEG* feature extractor perform in comparison to the generic feature extraction methods in terms of time complexity?

[22] https://www.virustotal.com/intelligence/help/file-search/

6. Does *MalJPEG,* when used with the classifier that provides the best detection results, provide better detection results than existing widely used antivirus engines?

### D. Evaluation Metrics

For evaluation purposes, we calculated the true positive rate (*TPR*) and the false positive rate (*FPR*) of each classifier. The *TPR* and *FPR* are the most important metrics in our domain of malware detection; a viable detection system must maintain a high *TPR* (representing the system's ability to successfully detect positive samples – malicious) and a low *FPR* (the system's capability of avoiding false alarms for negative samples – benign). Note that since our dataset is highly imbalanced, it is extremely important to use the *TPR* and *FPR* metrics instead of the well-known accuracy metric, as they represent the classifier's accuracy and false positives for the minority class only, i.e., malicious. Note also that the *TPR* and *FPR* are complementary to the true negative rate (*TNR*) and false negative rate (*FNR*).

In addition, we measured the area under the receiver operating characteristic (*ROC*) curve, or the *AUC*, of the machine learning classifiers presented in Section IV.C, trained and tested on different datasets. The *ROC* curve is created by plotting the true positive rate (*TPR*) against the false positive rate (*FPR*) at various threshold settings. In order to achieve a high *AUC*, a high *TPR* and low *FPR* are needed at each threshold. The *AUC* is a preferred metric (over the accuracy metric) for comparing machine learning algorithms applied on datasets (balanced or imbalanced, binary or multiclass) as suggested by [56], [57].

In order to calibrate the threshold (a value between zero and one) that optimizes the classifier's detection results, we needed a metric that integrates the *TPR* and *FPR*. Thus, we used the integrated detection rate (*IDR)* proposed by [3]. $IDR = TPR \cdot (1 - FPR) = TPR \cdot TNR$. As can be seen, the *IDR* integrates the *TPR* and *FPR* metrics. Since there is a trade-off between the *TPR* and *FPR* values, the *IDR* metric will have a maximum point at which the trade-off between the *TPR* and the *FPR* is optimal. The *IDR* measure makes it easier to identify the optimal point at which the *TPR* and *TNR* (1-*FPR*) are the highest. The well-known *F-measure* (or *F1*) metric is not adequate for our purpose of threshold calibration as it lacks the *FPR* metric. The *F-measure* is the harmonic mean of the *Precision* and *Recall* metrics; *Recall* is the same as the *TPR* metric, however *Precision* is not equal to the *FPR* metric. The *FPR* is a very important metric for malware detection systems as it represents the percentage of false alarms. A viable malware detection system must maintain a high *TPR* with a low *FPR*. The difference between using the *IDR* and *F-measure* for threshold calibration becomes very significant when coping with imbalanced datasets [3], such as the dataset used in this research.

### E. Experimental Design

Our experimental design is aimed at providing clear and practical answers to the abovementioned research questions.

The experiments were performed using the 10-fold cross-validation configuration (unless otherwise stated), and the fold results were grouped using average function. The cross-validation setup divides the dataset into training and test sets 10 times; each time the machine learning classifiers are trained on nine folds of the dataset and tested on the remaining fold. That ensures that the results reflect the detection of unknown malicious JPEG images.

Recall that since our collection is extremely imbalanced (just 1.15% malicious instances), the classification problem is much more difficult. In addition, classifiers which are susceptible to imbalanced datasets must be configured, and their threshold must be adjusted in order to achieve their optimal results in terms of *TPR* and *FPR*. We evaluated the classifiers with different threshold configurations and only present the results of the threshold which maximizes the *IDR* (combination of *TPR* and *FPR*).

#### 1) Experiment 1 – Comparison of Features Using Machine Learning Classifiers

In this experiment, we compare the detection results of the machine learning classifiers leveraging both *MalJPEG* features and features extracted using generic feature extraction methods.

First, we apply machine learning classifiers on datasets created using the *MalJPEG* feature extractor (which extracts the features described in Section IV.A) and different generic feature extraction methods with various configurations. Table 3 presents the configurations that we used for each of the feature extractors. With regard to the generic feature extraction methods, the specific configurations were chosen because previous studies indicated that they were well-suited for use with machine learning for malware detection. For example, the Min-Hash method was found to be effective when using 200 hash functions. We chose to examine different window sizes for different basic units (byte, char). The histogram method was found to work best with a window size = 1024 and stride = 256. Each unique configuration was used to produce a single dataset.

| Feature Extraction Method | Configuration | Number of features |
|---|---|---|
| Histogram | Basic Unit = Byte, Normalize = true | 256 |
| Histogram | Basic Unit = Char, Normalize = true | 128 |
| Entropy Histogram | Basic Unit = Byte, Window Size = 1024, Stride = 256, Byte Axis Size = 256, Entropy Axis Size = 8 | 2048 |
| Entropy Histogram | Basic Unit = Byte, Window Size = 1024, Stride = 256, Byte Axis Size = 16, Entropy Axis Size = 16 | 256 |
| Min-Hash | Basic Unit = Byte, Hash Functions = 200, Stride = 1, Window Size = {3,4,5,6,7,8,9,10,12,14,16,18,20,22,24,26, 28,30,32} | 200 |
| Min-Hash | Basic Unit = Char, Hash Functions = 200, Stride = 1, Window Size = {2,3,4,5,6,7,8,9,10} | 200 |
| *MalJPEG* | N/A | 10 |

**Table 3. Generic feature extraction methods used and their configurations.**

*2) Experiment 2 – Comparison of Feature Extraction Time Complexity*

In this experiment, we compare the *MalJPEG* feature extractor against the generic feature extraction methods in terms of time complexity. In order make this comparison, we applied all of the feature extractors on each file in our JPEG image collection (*156,818*) and measured the time it takes for the extractor to extract features from a single image. We compare the average time it takes for each extractor to extract features from files of different sizes.

In order to establish a fair benchmark, we warm up the Java virtual machine (JVM) so that all of the relevant classes are loaded in the cache, and thus can be accessed faster during runtime, equally for all feature extractors. We repeated this experiment five times and average the results.

*3) Experiment 3 – Comparison with Antivirus Engines*

In this experiment, we compare the *TPR* of leading antivirus engines with the *TPR* of the classifier that provided the best *AUC* in Experiment 1. We used the *VirusTotal* online Web service to obtain a comprehensive analysis of our malicious JPEG image collection, which contains 1,805 images, and their classification based on *VirusTotal's* 69 antivirus engines. We analyzed the report produced by *VirusTotal* and calculated the *TPR* for each engine by dividing the number of images that were recognized correctly as malicious by the total number of malicious images (1,805).

*4) Experiment 4 – Real-Life Unknown Malware Detection*

In this experiment, we tested *MalJPEG* in a real-life scenario in which the machine learning classifiers are tested only on more recently created files, i.e., instances that come later in time than the instances that the model was trained on.

The rationale behind this experiment is that in real-life, malicious files are created regularly, some of which are new and unknown in terms of the type of threat they pose, the vulnerability they exploit, their malicious way of action, etc. In this experiment, we want to demonstrate *MalJPEG's* detection capabilities on a dataset that represents this real-life scenario involving unknown instances.

We created a dataset with *MalJPEG* features and sorted the malicious instances based on the *"First Seen"* field of the *VirusTotal* report which denotes when the example was first introduced to *VirusTotal*. The First Seen field is our best indicator of the length of time a malicious JPEG image has been in the wild (i.e., the age of the malicious JPEG image). For the experiment, we derived five different datasets from the original sorted dataset, each of which uses a different percentage of instances to train the model: 50%, 66.6%, 75%, 80%, and 90%. For example, we trained machine learning classifiers on the first X% of the malicious and benign instances, and tested the model on the remaining instances. The lower the training percentage, the more difficult the experiment. In this experiment, we did not use the 10-fold cross-validation setup.

It is important to mention that it is likely that some of the "newer" files used to test the model are variants of "older" files used to train the model. Since variants of the same malware are assumed to be similar, is it possible that our model was already trained on samples that are similar to "newer" samples; thus, the "newer" samples are not always completely different from "older" ones, and there usually is some similarity between them. Therefore, the first experiment which applied 10-fold cross-validation is still relevant, and the fourth experiment adds a different perspective.

### F. Results

*1) Experiment 1*

Figure 7 presents a comparison between the detection results of the *Random Forest* classifier, evaluated on datasets created using the histogram methods presented in Table 3; we only provide the detection results of the *Random Forest* classifier, because it outperforms all of the other classifiers used in our experiments on all of the datasets created using the histogram methods. We set the *Random Forest* threshold to 0.05 (instead of the default of 0.5) to achieve the best results. The results are sorted from the highest to the lowest according to the *AUC* metric. As can be seen, the best results were achieved using the byte entropy histogram: *TPR* = 0.805, *FPR* = 0.05, *IDR* = 0.765, and *AUC* = 0.893.

Figure 8 presents a comparison between the detection results of the *K-Nearest Neighbors* classifier evaluated on datasets created using the Min-Hash methods. It is important to mention that the *K-Nearest Neighbors* classifier is the only classifier to use with Min-Hash datasets, since it is the only classifier that can actually compare Min-Hash signatures using a distance function (see Section IV.B.2); there is no actual order between the Min-Hash signature's numbers, thus regular machine learning algorithms are not effective on it. We used the *K-Nearest Neighbors* classifier with *K*=5 and distance function = *Hamming*. We set the *K-Nearest Neighbors* classifier threshold to 0.05 (instead of the default of 0.5) to achieve the highest results. The results are sorted from the highest to the lowest according to the *AUC* metric. As can be seen, the best results were achieved on the dataset created using Min-Hash based on ten-size byte shingles (*TPR* = 0.807, *FPR* = 0.064, *IDR* = 0.755, and *AUC* = 0.891), or on ten-size char shingles (*TPR* = 0.810, *FPR* = 0.054, *IDR* = 0.766, and *AUC* = 0.895).

Figure 9 presents the detection results of the machine learning classifiers applied on a dataset containing *MalJPEG* features. The optimal threshold (the one that maximizes the IDR) for the classifiers is 0.05. The results are sorted from the highest to the lowest according to the *AUC* metric. As can be seen, the *LightGBM* classifier achieved the highest *AUC = 0.997, with TPR* = 0.951, *FPR* = 0.04, and *IDR* = 0.948. These results answer the first and the second research questions and show that machine learning-based classifiers that have been trained on *MalJPEG* features can efficiently detect unknown malicious JPEG images.

Table 4 presents a summary of the configurations that provide the best results for all of the abovementioned methods, both generic feature extraction methods and the *MalJPEG* feature extractor. As can be seen, the best configurations for the histogram and Min-Hash methods provide nearly the same

results. However, the *LightGBM* classifier trained on *MalJPEG* features provides significantly better detection results in terms of the *TPR*, *FPR*, and *AUC*, with significantly less features required. It is important to clarify that the best detection results of the entropy histogram and Min-Hash methods (presented in Table 4) are not bad when considering that our dataset is extremely imbalanced and contains just 1.15% malicious instances. This also emphasizes the superior results obtained with *MalJPEG* features in comparison to features extracted using the best of the abovementioned generic feature extraction methods.

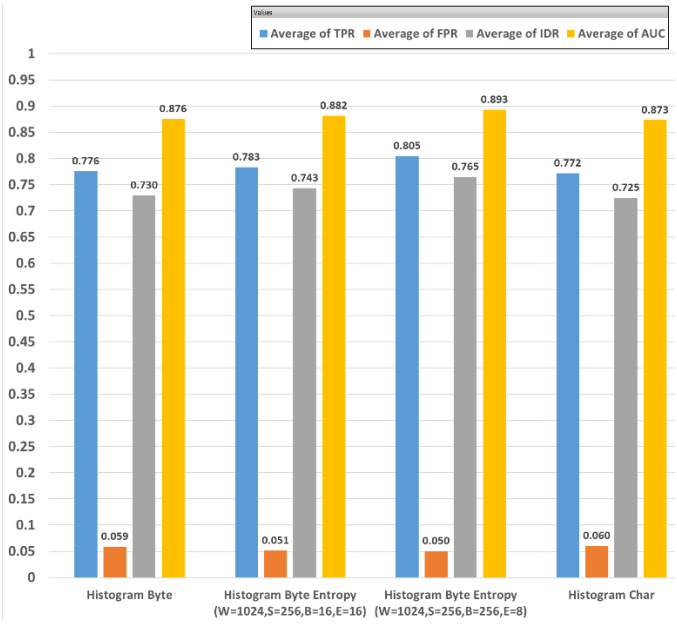


**Figure 7. Detection results for the *Random Forest* classifier on datasets created using different histogram feature extraction methods.**

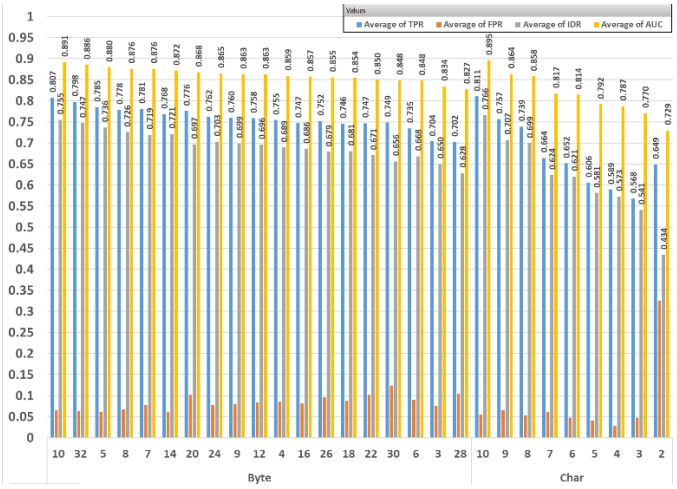


**Figure 8. Detection results for the *K-Nearest Neighbors* classifier on datasets created using *Min-Hash* feature extraction methods with different configurations.**

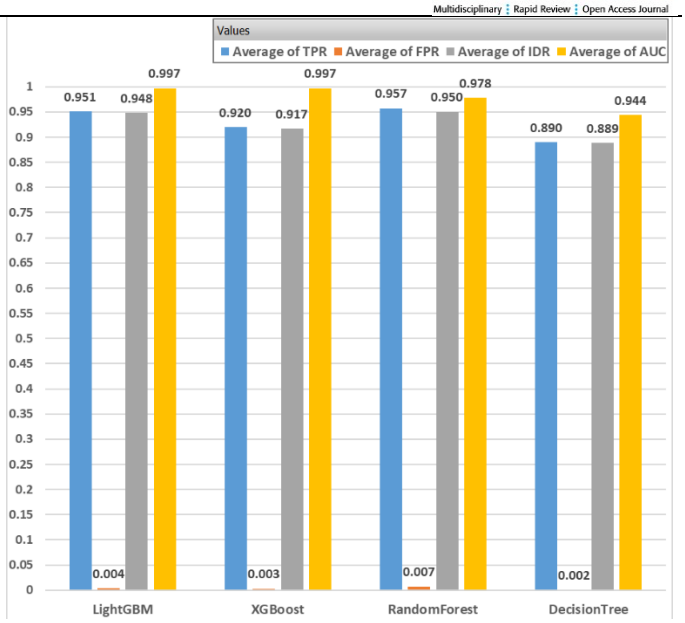


**Figure 9. Detection results of the machine learning classifiers on a dataset containing *MalJPEG* features.**

| Feature Extraction Method | Configuration | Number of features | Classifier | TPR | FPR | IDR | AUC |
|---|---|---|---|---|---|---|---|
| Entropy Histogram | Basic Unit = Byte, Window Size = 1024, Stride = 256, Bytes Axis Size = 256, Entropy Axis Size = 8 | 2048 | *Random Forest* | 0.805 | 0.050 | 0.765 | 0.893 |
| Min-Hash | Basic Unit = Byte, Hash Functions = 200, Window Size = 10, Stride = 1 | 200 | *Random Forest* | 0.810 | 0.054 | 0.766 | 0.895 |
| *MalJPEG* | N/A | 10 | *LightGBM* | **0.951** | **0.004** | **0.948** | **0.997** |

**Table 4. Summary of the configurations that provide the best results for both histogram and *Min-Hash* methods.**

### *2) Experiment 2*

Figure 10 presents the average time (in milliseconds) required for each feature extractor to extract features from images in our collection (the y-axis presents the time in milliseconds, and the x-axis presents the file size in KB). As can be seen, as the file size grows, it takes more time for all of the feature extractors to extract the features. One can also see that 1) the slowest feature extractor is Min-Hash (dark blue); 2) the byte entropy histograms behave almost the same (gray and yellow); and 3) the *MalJPEG* feature extractor (green) is the fastest, and it behaves similarly to the byte and char histogram methods (orange and light blue). Table 5 contains the values presented in Figure 10 for the following file size ranges: 0-1000KB and 7500-9600KB.

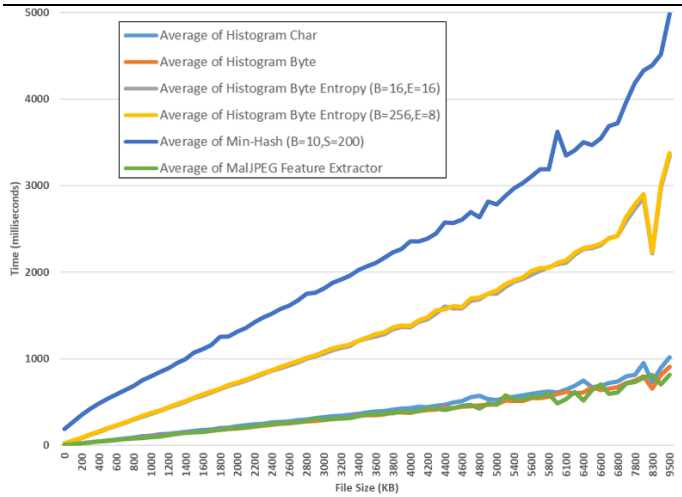


**Figure 10. The average time required for each feature extractor to extract features from JPEG images in our image collection based on the file size.**

| File Size (KB) | MalJPEG Feature Extractor | Histogram Byte | Histogram Char | Histogram Byte Entropy (B=16, E=16) | Histogram Byte Entropy (B=256, E=8) | Min-Hash Byte (W=10) |
|---|---|---|---|---|---|---|
| [0-100] | 6 | 6 | 6 | 18 | 22 | 191 |
| [100-200] | 15 | 15 | 16 | 51 | 55 | 271 |
| [200-300] | 24 | 24 | 26 | 85 | 90 | 351 |
| [300-400] | 34 | 34 | 37 | 122 | 127 | 422 |
| [400-500] | 44 | 44 | 47 | 158 | 163 | 483 |
| [500-600] | 52 | 54 | 59 | 196 | 201 | 540 |
| [600-700] | 62 | 64 | 69 | 228 | 232 | 588 |
| [700-800] | 71 | 74 | 80 | 260 | 267 | 637 |
| [800-900] | 77 | 82 | 92 | 296 | 303 | 690 |
| [900-1000] | 85 | 93 | 105 | 333 | 340 | 756 |
| … | … | … | … | … | … | … |
| [7500-7600] | 718 | 719 | 796 | 2593 | 2640 | 3968 |
| [7800-7900] | 734 | 750 | 813 | 2734 | 2781 | 4188 |
| [8200-8300] | 782 | 797 | 953 | 2860 | 2906 | 4328 |
| [8300-8400] | 635 | 657 | 734 | 2219 | 2234 | 4391 |
| [8400-8500] | 703 | 813 | 898 | 2977 | 3009 | 4516 |
| [9500-9600] | 813 | 906 | 1016 | 3343 | 3375 | 4984 |

**Table 5. The average time required for each feature extractor to extract features from JPEG images in our collection based on the file size.**

### 3) *Experiment 3*

Figure 11 presents a comparison between the *TPR* achieved by the *LightGBM* classifier (achieved the best results in Experiment 1) trained on *MalJPEG* features and the top 12 antivirus engines out of VirusTotal's 69 antivirus engines. As can be seen, our method significantly outperforms all of the leading antivirus engines. Our method achieved a *TPR* of 0.951, while the most accurate antivirus, *Fortinet*, had a *TPR* of 0.823; therefore, our method is ~15.5% better at the task of malicious JPEG image detection than *Fortinet*. It is important to mention that the average *TPR* of the top 12 antivirus engines (0.73) is relatively low in comparison to the average *TPR* of the classifiers we used in the previous experiment (0.929).

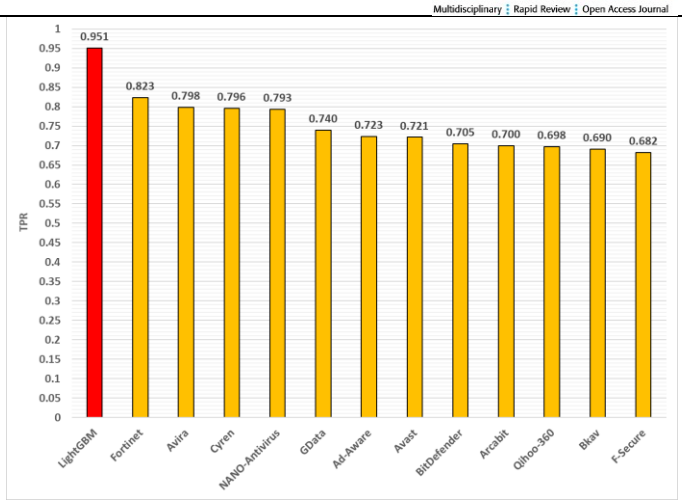


**Figure 11. The TPR for the *LightGBM* classifier obtained in Experiment 1 compared to 12 top antivirus engines.**

### 4) *Experiment 4*

Figure 12 presents the detection results of the *LightGBM* classifier which provided the best detection results on the five real-world datasets used in this experiment. The results are sorted from the lowest to the highest according to the *AUC* metric. The *LightGBM* classifier achieved *AUCs* between 0.975 and 0.996, *TPRs* between 0.865 and 0.911, *FPRs* between 0.001 and 0.09, and *IDRs* between 0.864 and 0.903. As in the first experiment, the optimal threshold for the *LightGBM* classifier is 0.05. In general, the detection results achieved by each of the classifiers, and *LightGBM* in particular, are slightly lower (in terms of the *AUC* and *TPR*) than the corresponding results of the same classifiers in Experiment 1. Yet, these results still demonstrate that machine learning-based classifiers trained on *MalJPEG* features can efficiently detect unknown malicious JPEG images in a real-life scenario in which the classifier is tested only on future unknown instances.

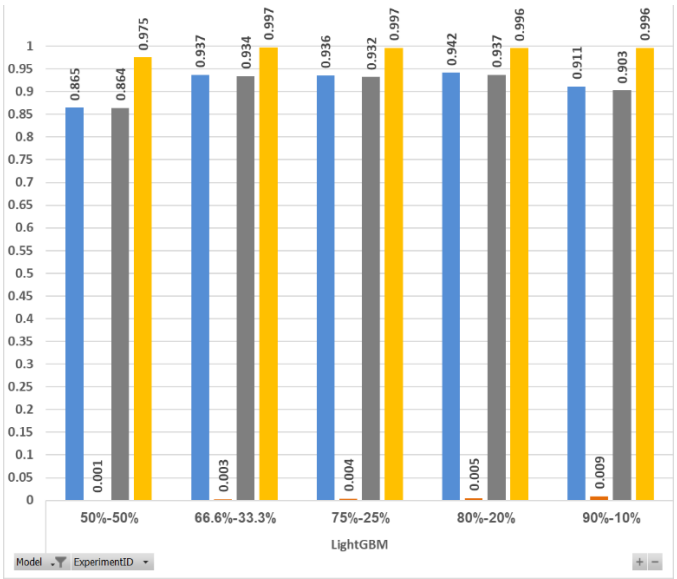


**Figure 12. Detection results of the *LightGBM* classifiers on five real-life datasets (%train-%test) containing *MalJPEG* features.**

To further analyze the results, we examined the properties of misclassified malicious samples (false negatives), which totaled 16, in order to understand why they were misclassified. We tried to determine whether they have commonalities and if they differ from the samples which were classified correctly (true positives). We found that 13 of the 16 samples contain a Trojan with a backdoor labeled as *'PHP/Agent.VD!tr.bd'*, two contain an exploit (*CVE-2010-0028* and *CVE-2012-0897*), and one contain a ransomware labeled as *'W32/Ransom.AAN!tr'* (the labels and CVEs provided by the *Fortinet* antivirus engine). However, only the samples which contain exploits are unique to the misclassified samples; the remaining 14 samples contain threats that are also found in the true positive samples. In addition, we did not find any characteristics in the features of the 16 misclassified samples which differentiate them from the true positive samples and could cause the classifier to misclassify them.

## VI. Discussion and Conclusion

In this paper, we present *MalJPEG*, a machine learning-based solution for efficient detection of unknown malicious JPEG images. To the best of our knowledge, we are the first to present a machine learning-based solution tailored specifically for the detection of malicious JPEG images. *MalJPEG* extracts 10 simple but discriminative features from the JPEG file structure and leverages them with a machine learning classifier, in order to discriminate between benign and malicious JPEG images.

*MalJPEG* features are extracted based on the structure of the JPEG image. *MalJPEG* features were defined based on an understanding of how attackers use JPEG images in order to launch attacks and how it affects the JPEG file structure in comparison to regular benign JPEG images. The features are simple and relatively easy to extract statically (without actually viewing the image) when parsing the JPEG image file.

We evaluate *MalJPEG* in four experiments. For our evaluation, we used a very large collection of 156,818 JPEG images: 155,013 (98.9%) benign and 1,805 (1.15%) malicious, collected between 2016 and 2018 from social media (benign images) and *VirusTotal* (malicious images). Note that the percentage of malicious images in our collection is extremely low (1.15%). We intentionally prepared our collection that way so the collection reflects, as much as possible, the low percentage of malicious images (compared to benign images) in the real world. Note also that the extremely low percentage of malicious instances (positive) in the collection makes the detection of malicious images in our experiments extremely difficult.

In the first experiment, we compared the detection results of machine learning classifiers evaluated on datasets created using two generic feature extraction methods proposed in previous studies, (histogram and Min-Hash), with various configurations, against *MalJPEG features*. The results showed that the best configurations of the histogram (byte entropy histogram) and Min-Hash methods provide roughly the same results using the *Random Forest* classifier: $TPR = 0.810$, $FPR = 0.054$, and $AUC = 0.895$. However, the results of this experiment also showed that the *LightGBM* classifier trained on *MalJPEG* features provides improved results: $TPR = 0.951$, $FPR = 0.004$, and $AUC = 0.997$. It is important to emphasize that *MalJPEG* is capable of providing these significantly better detection results using only 10 features, in comparison to the Min-Hash and histogram methods, which used 200 and 2048 features respectively. These results also prove that *MalJPEG* features are discriminative, thus allowing machine learning classifiers to distinguish between malicious and benign JPEG images, although the dataset is highly imbalanced.

In the second experiment, we compared the time complexity of the *MalJPEG* feature extractor to the generic feature extraction methods, for all of the files in our image collection. The results showed that the *MalJPEG* feature extractor is the fastest (24ms for an average file of 200-300KB) and outperforms all of the other feature extractors.

In the third experiment, we compared *MalJPEG* based on the *LightGBM* classifier (which achieved the best detection results in Experiment 1), to 12 leading antivirus engines. The results showed that the *TPR* of the *LightGBM* classifier significantly outperforms all of the top 12 antivirus engines in the task of malicious JPEG image detection. The *LightGBM* classifier is 15.5% better than the most successful antivirus engine in this experiment, *Fortinet*, which achieved a *TPR* of 0.823. Bearing in mind that our malicious JPEG collection only contains known threats collected from *VirusTotal*, this experiment clearly demonstrates the inability of even the leading antivirus engines to detect malicious JPEG images containing known threats. Antivirus engines which are based solely on signatures can only detect known threats if they are constantly and quickly updated with their signatures. In contrast, *MalJPEG* which is machine learning-based, is able to effectively detect both known and unknown malicious JPEG images.

In the fourth experiment, we tested *MalJPEG* on five datasets which represent a real-life scenario in which we trained the machine learning model on prior instances (chronologically) and tested the model only on unknown future instances. The datasets differ from each other by the percentage of instances used for model training (50%, 66.6%, 75%, 80%, and 90%); the lower the training percentage, the harder the experiment. The best detection results were achieved by the *LightGBM* classifier, which obtained *AUCs* between 0.975 and 0.996, *TPRs* between 0.865 and 0.911, *FPRs* between 0.001 and 0.09, and *IDRs* between 0.864 and 0.903. These detection results are slightly worse than the corresponding results obtained in Experiment 1, yet they still demonstrate that a machine learning-based classifier trained on *MalJPEG* features can also efficiently detect unknown malicious JPEG images in a real-life scenario in which the classifier is tested on future unknown instances.

Given the threats posed against individuals, businesses, and organizations by cyber attackers using malicious JPEG images, a comprehensive detection method is clearly required. *MalJPEG* provides efficient detection of known and unknown malicious JPEG images. *MalJPEG* works relatively fast, thus

supporting real-time performance requirements for the detection of large image streams. In addition, *MalJPEG* can be parallelized easily and scaled to cope with the massive amount of images in the large-scale systems of enterprises. Based on the results of our experiments, it would be valuable to implement *MalJPEG,* in order to protect organizations, cloud services (e.g., *Microsoft Office 365*, *Google Drive*, etc.), social media (*Facebook*, *Instagram*, etc.), and their users from malicious JPEG images.

## VII. Acknowledgements

We would like to thank *VirusTotal* for granting us access to their malware collection for academic use.

## VIII. References

[1] A. Cohen, N. Nissim, L. Rokach, and Y. Elovici, "SFEM: Structural feature extraction methodology for the detection of malicious office documents using machine learning methods," *Expert Syst. Appl.*, vol. 63, pp. 324–343, Nov. 2016.

[2] N. Nissim, A. Cohen, and Y. Elovici, "ALDOCX: Detection of Unknown Malicious Microsoft Office Documents Using Designated Active Learning Methods Based on New Structural Feature Extraction Methodology," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 3, pp. 631–646, Mar. 2017.

[3] A. Cohen, N. Nissim, and Y. Elovici, "Novel set of general descriptive features for enhanced detection of malicious emails using machine learning methods," *Expert Syst. Appl.*, vol. 110, pp. 143–169, Nov. 2018.

[4] N. Nissim, A. Cohen, C. Glezer, and Y. Elovici, "Detection of malicious PDF files and directions for enhancements: A state-of-the art survey," *Comput. Secur.*, vol. 48, pp. 246–266, 2015.

[5] N. Nissim, Y. Lapidot, A. Cohen, and Y. Elovici, "Trusted system-calls analysis methodology aimed at detection of compromised virtual machines using sequential mining," *Knowledge-Based Syst.*, vol. 153, pp. 147–175, 2018.

[6] A. Cohen and N. Nissim, "Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory," *Expert Syst. Appl.*, vol. 102, pp. 158–178, Jul. 2018.

[7] N. Nissim, A. Cohen, R. Moskovitch, A. Shabtai, M. Edri, O. BarAd, and Y. Elovici, "Keeping pace with the creation of new malicious PDF files using an active-learning based detection framework," *Secur. Inform.*, vol. 5, no. 1, p. 1, 2016.

[8] N. Nissim, Y. Lapidot, A. Cohen, and Y. Elovici, "Trusted system-calls analysis methodology aimed at detection of compromised virtual machines using sequential mining," *Knowledge-Based Syst.*, vol. 153, pp. 147–175, Aug. 2018.

[9] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, "Novel active learning methods for enhanced PC malware detection in windows OS," *Expert Syst. Appl.*, vol. 41, no. 13, pp. 5843–5857, 2014.

[10] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, "Detecting unknown computer worm activity via support vector machines and active learning," *Pattern Anal. Appl.*, vol. 15, no. 4, pp. 459–475, Nov. 2012.

[11] D. Nahmias, A. Cohen, N. Nissim, and Y. Elovici, "TrustSign: Trusted Malware Signature Generation in Private Clouds Using Deep Feature Transfer Learning," 2019, pp. 1–8.

[12] S. Sharma, C. Rama Krishna, and S. K. Sahay, "Detection of advanced malware by machine learning techniques," in *Advances in Intelligent Systems and Computing*, 2019, vol. 742, pp. 333–342.

[13] T. Kumar, S. Sharma, H. Goel, S. Chaudhary, and P. Jain, "A Novel Machine Learning Approach for Malware Detection," *SSRN Electron. J.*, 2019.

[14] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers and Security*, vol. 81. pp. 123–147, 2019.

[15] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—A state of the art survey," *ACM Comput. Surv.*, vol. 52, no. 5, 2019.

[16] N. Nissim, A. Cohen, R. Moskovitch, A. Shabtai, M. Edry, O. Bar-Ad, and Y. Elovici, "ALPD: Active learning framework for enhancing the detection of malicious PDF files," in *Proceedings - 2014 IEEE Joint Intelligence and Security Informatics Conference, JISIC 2014*, 2014, pp. 91–98.

[17] N. Nissim, A. Cohen, J. Wu, A. Lanzi, L. Rokach, Y. Elovici, and L. Giles, "Scholarly digital libraries as a platform for malware distribution," *Cryptol. Inf. Secur. Ser.*, vol. 15, pp. 107–128, 2017.

[18] N. Nissim, A. Cohen, J. Wu, A. Lanzi, L. Rokach, Y. Elovici, and L. Giles, "Sec-Lib: Protecting Scholarly Digital Libraries from Infected Papers Using Active Machine Learning Framework.," *IEEE Access*, pp. 1–1, 2019.

[19] Y. S. Jeong, J. Woo, and A. R. Kang, "Malware Detection on Byte Streams of PDF Files Using Convolutional Neural Networks," *Secur. Commun. Networks*, vol. 2019, 2019.

[20] N. Nissim, A. Cohen, and Y. Elovici, "Boosting the detection of malicious documents using designated active learning methods," *Proc. - 2015 IEEE 14th Int. Conf. Mach. Learn. Appl. ICMLA 2015*, pp. 760–765, 2016.

[21] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[22] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *ASIA CCS 2017 -*

*Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, 2017, pp. 506–519.

[23] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[24] D. Hendrycks and K. Gimpel, "Early methods for detecting adversarial images," in *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, 2019.

[25] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, "A study of the effect of JPG compression on adversarial images," Aug. 2016.

[26] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, 2019.

[27] R. S. Kunwar and P. Sharma, "Framework to detect malicious codes embedded with JPEG images over social networking sites," in *Proceedings of 2017 International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2017*, 2018, vol. 2018–Janua, pp. 1–4.

[28] C. Y. Lin and S. F. Chang, "A robust image authentication method distinguishing JPEG compression from malicious manipulation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 2, pp. 153–168, 2001.

[29] T. Denemark, P. Bas, and J. Fridrich, "Natural Steganography in JPEG Compressed Images," *Electron. Imaging*, vol. 2018, no. 7, pp. 316-1-316–10, Jan. 2018.

[30] X. Hu, J. Ni, and Y.-Q. Shi, "Efficient JPEG Steganography Using Domain Transformation of Embedding Entropy," *IEEE Signal Process. Lett.*, vol. 25, no. 6, pp. 773–777, Jun. 2018.

[31] Z. Bao, X. Luo, Y. Zhang, C. Yang, and F. Liu, "A Robust Image Steganography on Resisting JPEG Compression with No Side Information," *IETE Tech. Rev.*, pp. 1–10, Jun. 2018.

[32] T. Denemark and J. Fridrich, "Steganography With Multiple JPEG Images of the Same Scene," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 10, pp. 2308–2319, Oct. 2017.

[33] M. Chen, V. Sedighi, M. Boroumand, and J. Fridrich, "JPEG-Phase-Aware Convolutional Neural Network for Steganalysis of JPEG Images," in *Proceedings of the 5th ACM Workshop on Information Hiding and Multimedia Security - IHMMSec '17*, 2017, pp. 75–84.

[34] C. Chen and Y. Q. Shi, "JPEG image steganalysis utilizing both intrablock and interblock correlations," in *Proceedings - IEEE International Symposium on Circuits and Systems*, 2008.

[35] J. Yang, Y.-Q. Shi, E. K. Wong, and X. Kang, "JPEG Steganalysis Based on DenseNet," Nov. 2017.

[36] X. Wu, Z. Shao, P. Ou, and S. Tan, "Application of quantisation-based deep-learning model compression in JPEG image steganalysis," *J. Eng.*, vol. 2018, no. 16, pp. 1402–1406, Nov. 2018.

[37] A. Zakaria, M. Chaumont, and G. Subsol, "Quantitative and Binary Steganalysis in JPEG: A Comparative Study," Sep. 2018.

[38] R. Shin and D. Song, "JPEG-resistant Adversarial Images," *Proc. Mach. Learn. Comput. Secur. Work.*, 2017.

[39] A. Prakash, N. Moran, S. Garber, A. Dilillo, and J. Storer, "Protecting JPEG images against adversarial attacks," in *Data Compression Conference Proceedings*, 2018, vol. 2018–March, pp. 137–146.

[40] Y. Luo, H. Zi, Q. Zhang, and X. Kang, "Anti-forensics of JPEG compression using generative adversarial networks," in *European Signal Processing Conference*, 2018, vol. 2018–Septe, pp. 952–956.

[41] N. Das, M. Shanbhogue, S.-T. Chen, F. Hohman, L. Chen, M. E. Kounavis, and D. H. Chau, "Keeping the Bad Guys Out: Protecting and Vaccinating Deep Learning with JPEG Compression," May 2017.

[42] A. E. Aydemir, A. Temizel, and T. T. Temizel, "The Effects of JPEG and JPEG2000 Compression on Attacks using Adversarial Examples," Mar. 2018.

[43] C. Lee and G. G. Lee, "Information gain and divergence-based feature selection for machine learning-based text categorization," *Inf. Process. Manag.*, vol. 42, no. 1 SPEC. ISS, pp. 155–165, Jan. 2006.

[44] E. M. Rudd, R. Harang, and J. Saxe, "MEADE: Towards a Malicious Email Attachment Detection Engine," Apr. 2018.

[45] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software, MALWARE 2015*, 2016, pp. 11–20.

[46] O. Chum, J. Philbin, and A. Zisserman, "Near Duplicate Image Detection: min-Hash and tf-idf Weighting," in *Procedings of the British Machine Vision Conference 2008*, 2008, p. 50.1-50.10.

[47] W. Jin, S. Chaki, C. Cohen, A. Gurfinkel, J. Havrilla, C. Hines, and P. Narasimhan, "Binary function clustering using semantic hashes," in *Proceedings - 2012 11th International Conference on Machine Learning and Applications, ICMLA 2012*, 2012, vol. 1, pp. 386–391.

[48] J. Crussell, C. Gibler, and H. Chen, "Scalable

Semantics-Based Detection of Similar Android Applications," in *Esorics*, 2013, pp. 182–199.

[49] A. Akusok, Y. Miche, J. Hegedus, R. Nian, and A. Lendasse, "A Two-Stage Methodology Using K-NN and False-Positive Minimizing ELM for Nominal Data Classification," *Cognit. Comput.*, vol. 6, no. 3, pp. 432–445, 2014.

[50] A. Tamersoy, K. Roundy, and D. H. Chau, "Guilt by association," *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '14*, pp. 1524–1533, 2014.

[51] C. Oprisa, M. Checiches, and A. Nandrean, "Locality-sensitive hashing optimizations for fast malware clustering," in *Proceedings - 2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing, ICCP 2014*, 2014, pp. 97–104.

[52] N. Srndic and P. Laskov, "Detection of Malicious PDF Files Based on Hierarchical Document Structure," *Proc. 20th Annu. Netw. Distrib. Syst. Symp.*, 2013.

[53] N. Šrndić and P. Laskov, "Hidost: a static machine-learning-based detector of malicious files," *Eurasip J. Inf. Secur.*, vol. 2016, no. 1, 2016.

[54] M. Ahmadi, A. Sotgiu, and G. Giacinto, "IntelliAV: Toward the feasibility of building intelligent anti-malware on android devices," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, vol. 10410 LNCS, pp. 137–154.

[55] R. Moskovitch, D. Stopel, C. Feher, N. Nissim, N. Japkowicz, and Y. Elovici, "Unknown malcode detection and the imbalance problem," *J. Comput. Virol.*, vol. 5, no. 4, pp. 295–308, 2009.

[56] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 3, pp. 299–310, 2005.

[57] C. X. Ling, J. Huang, and H. Zhang, "AUC: A better measure than accuracy in comparing learning algorithms," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2003, vol. 2671, pp. 329–341.

**AVIAD COHEN**

Aviad Cohen is a security researcher at the Malware-Lab, Cyber Security Research Center (CSRC) at Ben-Gurion University of the Negev. Aviad pursue his Ph.D. studies between 2015 and 2019 in BGU's Department of Software and Information Systems Engineering. His research is aimed at the development of a triple-layered machine learning-based methodology for enhancing the security of E-mail ecosystem. He is a co-author of several papers dealing with the analysis and detection of malicious non-executable files, malicious emails and compromised virtual machine. His main areas of interest are cyber security, machine learning and data science.

**DR. NIR NISSIM**

Nir Nissim is a researcher and the Head of the Malware Lab at the Cyber Security Research Center at Ben-Gurion University. In 2018 Dr. Nissim has joined the department of Industrial Engineering and Management at Ben-Gurion University. Prior to this, in 2016, Dr. Nissim completed his Ph.D. with honors at BGU's Department of Information Systems Engineering; he received a prestigious prize in recognition of his ranking as the Faculty of Engineering Sciences' top doctoral student in 2016. During his Ph.D. research, Dr. Nissim won several best paper awards in top ranked scientific international conferences and awards of excellence at BGU, and he was one of the few doctoral students at BGU to win an exclusive doctoral cyber security scholarship granted by the Israeli Cyber Security Bureau. Dr. Nissim was a postdoctoral fellow at Stanford University and served as a data-scientist in a neuroscience research dealing with analyzing spike-level data of primates. Dr. Nissim published several noteworthy papers dealing with the development of a generic active learning framework aimed at the detection and acquisition of various types of malware in a variety of platforms. Dr. Nissim is recognized as an expert in information systems security and machine learning solutions and has been leading several large-scale research projects in the field, including collaborative projects between academia and industry. His main areas of interests are mobile and computer security, security of medical devices, machine learning, and data science. In addition to his contributions to the cyber security domain, Dr. Nissim is also interested in the bioinformatics domain and has published a number of papers regarding the efficient classification of condition severity. Dr. Nissim is also a lecturer on cyber security and machine learning topics, both in the Information Systems Engineering Department at BGU, as well as in the Industrial Engineering and Management Department at Tel Aviv University. In addition, Dr. Nissim is the Head of the ICSML program which is an international cyber-security and Machine learning academic and professional program for international students.

**PROF. YUVAL ELOVICI**

Yuval Elovici is the director of the Telekom Innovation Laboratories at Ben-Gurion University of the Negev (BGU), head of BGU Cyber Security Research Center, Professor in the Department of Software and Information Systems Engineering at BGU. He holds B.Sc. and M.Sc. degrees in Computer and Electrical Engineering from BGU and a Ph.D. in Information Systems from Tel-Aviv University. His primary research interests are computer and network security, cyber security, web intelligence, information warfare, social network analysis, and machine learning. Prof. Elovici also consults professionally in the area of cyber security and is the co-founder of Morphisec, startup company that develop innovative cyber-security mechanisms that relate to moving target defense.