The detection of malicious JPEG images

December 2021

1 Introduction

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. In this paper[1], Aviad Cohen, Nir Nissim and Yuval Elovici 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.

Ran Dubin provided us with 1,805 malicious images for this project. In addition, we found and downloaded 27,364 benign images. Images were excluded from the benign dataset if they alerted the antivirus or couldn't be decoded. Importantly, A. Cohen, N. Nissim, and Y. Elovichi 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.

2 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* and *MyMalJPEG* solutions presented in this research. The format of JPEG images is comprehensively described in the JPEG File Interchange Format (JFIF) specification.

2.1 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 o 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 marke'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 *OI*, and the name of **0xFFFE** is *COM*. Segment types **0xFF01** and **0xFFD8 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. Figure 1 presents the possible markers, their hexadecimal code, and their definition/purpose.

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 **JP**E**G** extensions
JPG_n	0xFFF0-0xFFFD	Reserved for **JP**E**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 ***F**rame*
SOS	0xFFDA	**S**tart **o**f ***S**can*
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

Figure 1: Possible JPEG Markers

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 2 presents the hexadecimal view of a sample JPEG image file.

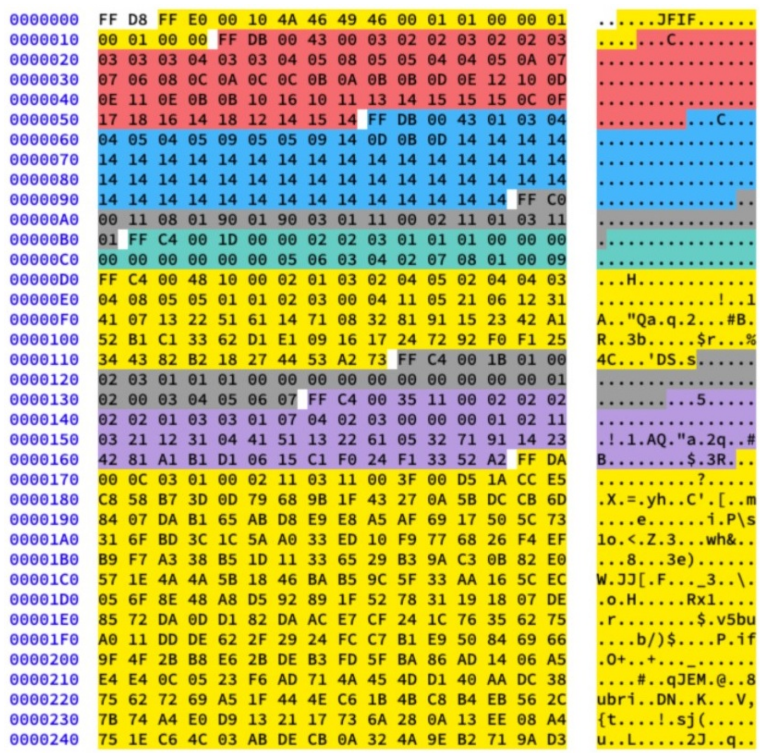

Figure 2: JPEG file structure in hexadecimal view

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.

Figure 3 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.

2.2 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

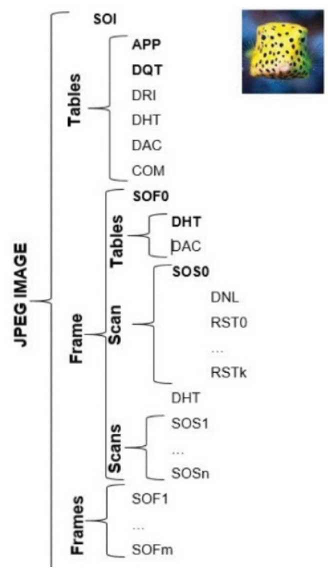

Figure 3: The structure of a simple JPEG image and the hierarchy of the markers and their division into frames and scans. The markers in bold are mandatory or the most common markers

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.

Steganography (steganos – covered, graphie – writing) – Steganography (Figure 4), 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 work, we focus only on the later and the detection of malicious JPEG images, and not on steganography detection.

Figure 4: The Zeus banking Trojan (ZeusVM)

3 Methods

3.1 MalJPEG Solution

A. Cohen, N. Nissim, and Y. Elovichi present the compact set of discriminative features extracted by *MalJPEG*. They engineered these features after manually examining the structure of many benign and malicious JPEG images. They gained an understanding of how attackers use JPEG images in order to launch attacks and how it affects the JPEG file structure. They 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, they 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. Figure 5 contains the set of *MalJPEG* features; Note that, all of the features are numeric.

Figure 6 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 $TPR = 0.951$,

#	Feature Name	Description
1	Marker_EOI_content _after_num	Number of bytes after the EOI (end of file) marker.
2	Marker_DHT_size_max	Maximal DHT marker size found in the file.
3	File_size	Image file size in bytes.
4	Marker_APP1_size_max	Maximal APP1 marker size found in the file.
5	Marker_COM_size_max	Maximal COM marker size found in the file.
6	Marker_DHT_num	Number of DHT markers found in the file.
7	File_markers_num	Total number of markers found in the file.
8	Marker_DQT_num	Number of DQT markers found in the file.
9	Marker_DQT_size_max	Maximal DQT marker size found in the file.
10	Marker_APP12_size_max	Maximal APP12 marker size found in the file.

Figure 5: MalJPEG Features

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.

Figure 7 presents a comparison between the *TPR* achieved by the *LightGBM* classifier trained on *MalJPEG* features and the top 12 antivirus engines out of VirusTotal's 69 antivirus engines. As can be seen, their 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).

3.2 MyMalJPEG Solution

Feature Name	Description	Source

Marker_EOI_content_after_num	Number of bytes after the EOI (end of file) marker	MalJPEG
File_markers_num	Total number of markers found in the file	MalJPEG
File_size	Image file size in bytes	MalJPEG
Marker_APP1_size_max	Maximal APP1 marker size found in the file	MalJPEG
Marker_APP12_size_max	Maximal APP12 marker size found in the file	MalJPEG
Marker_COM_size_max	Maximal COM marker size found in the file	MalJPEG
Marker_DHT_num	Number of DHT markers found in the file	MalJPEG
Marker_DHT_size_max	Maximal DHT marker size found in the file	MalJPEG
Marker_DQT_num	Number of DQT markers found in the file	MalJPEG
Marker_DQT_size_max	Maximal DQT marker size found in the file	MalJPEG
Marker_APP_other_size_max	Maximal APP0-12 markers size found in the file	JPEG markers
Marker_APP_other_num	Number of APP0-12 markers found in the file	JPEG markers
Marker_SOF_size_max	Maximal SOF marker size found in the file	JPEG markers
Marker_DRI_num	Number of DRI markers found in the file	JPEG markers
Marker_RST_num	Number of RST markers found in the file	JPEG markers
Marker_SOS_len	Length of SOS segment marker found in the file	JPEG markers
Marker_EOI	Flag for EOI marker found in the file	JPEG markers
ExifInteroperabilityOffset	This is a byte offset into the Interoperability IFD "table"	EXIF Tags
SceneCaptureType	This tag indicates the type of scene that was shot. It can also be used to record the mode in which the image was shot. Note that this differs from the SceneType tag	EXIF Tags
MeteringMode	The metering mode	EXIF Tags
LightSource	The kind of light source	EXIF Tags

Flash	Indicates the status of flash when the image was shot	EXIF Tags
ColorSpace	This tag specifies the color space in which the rendered preview in this IFD is stored. The default value for this tag is sRGB for color previews and Gray Gamma 2.2 for monochrome previews	EXIF Tags
Make	The manufacturer of the recording equipment. This is the manufacturer of the DSC, scanner, video digitizer or other equipment that generated the image. When the field is left blank, it is treated as unknown	EXIF Tags
ExposureProgram	The class of the program used by the camera to set exposure when the picture is taken	EXIF Tags
ResolutionUnit	The unit for measuring XResolution and YResolution. The same unit is used for both XResolution and YResolution. If the image resolution is unknown, 2 (inches) is designated	EXIF Tags
Exception_num	Number of exceptions caught during tag reading	EXIF Tags

Table 1: MyMalJPEG Features

We present the compact set of discriminative features extracted from *JPEG* image. We engineered these features after manually examining the structure of *JPEG* markers. 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 1 contains the set of *MyMalJPEG* features; Note that, all of the features are numeric.

MalJPEG gave special attention to APP1, therefore we decided to determine what kind of data might be there. If the data has been changed, we'll be able to catch it. *APP1* includes *EXIF* metadata, TIFF IFD format, JPEG thumbnail, and Adobe XMP. We focused on EXIF metadata, *EXIF* stands for "Exchangeable Image File Format". This type of information is formatted according to the TIFF specification, and may be found in JPG images. The *EXIF* meta information is organized into different Image File Directories (IFD's) within an image. The names of these IFD's correspond to the ExifTool family 1 group names. *Exif* JPEG file uses an *APP1* segment to store the information

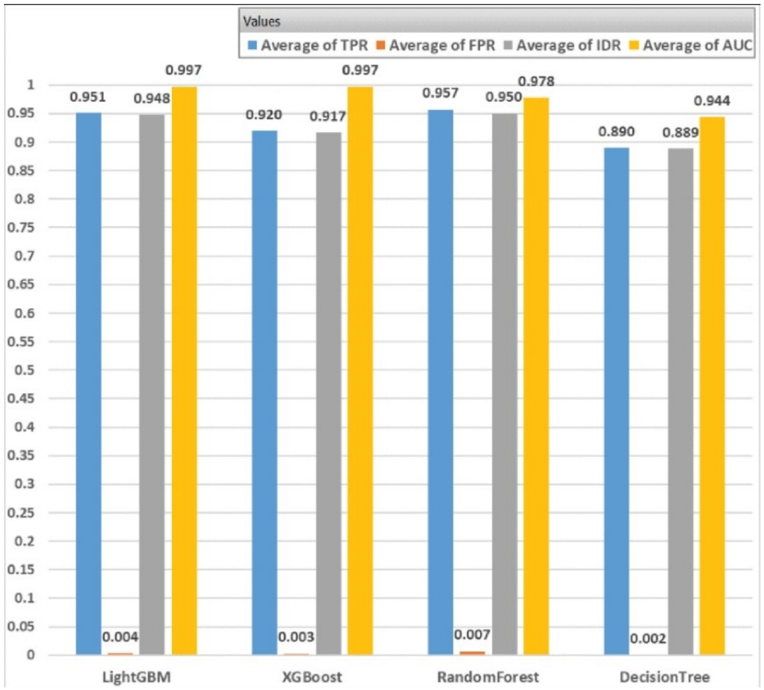

Figure 6: Detection results of the machine learning classifiers on a dataset containing MalJPEG features

(and multiples *APP2* segments for *flashPix* data). *Exif APP1* segment stores a great amount of information on photographic parameters for digital cameras and it is the preferred way to store thumbnail images nowadays. It can also host an additional section with GPS data. In theory, *Exif APP1* is recorded immediately after the *SOI* marker (the marker indicating the beginning of the file).

Figure 8 presents the detection results of the machine learning classifiers applied on a dataset containing *MyMalJPEG* features. It is clear that our improvements to *MalJPEG* resulted in better results.

4 Histogram

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. A. Cohen, N. Nissim and Y. Elovici presented 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

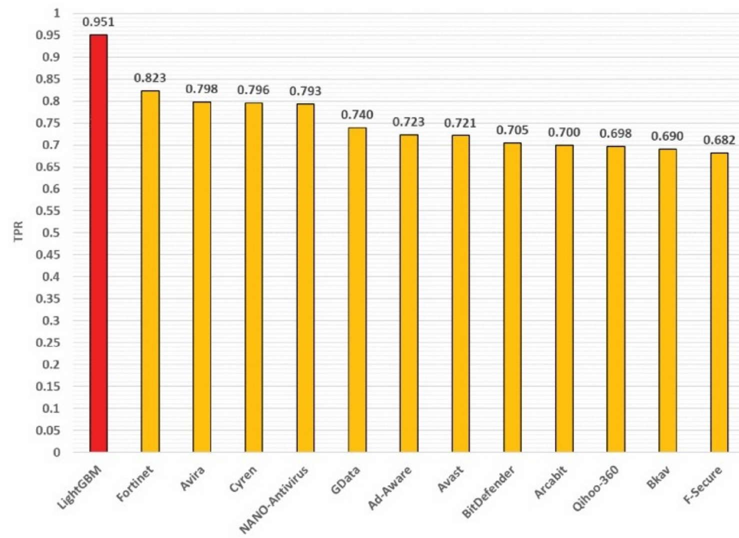

Figure 7: The TPR for the LightGBM classifier obtained in Experiment 1 compared to 12 top antivirus engines

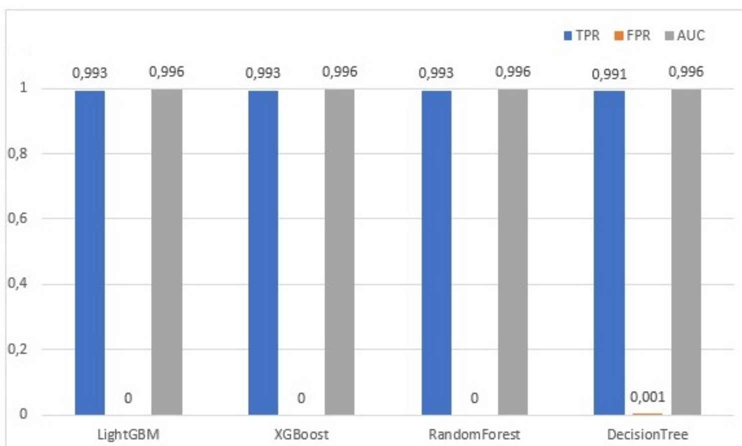

Figure 8: Detection results of the machine learning classifiers on a dataset containing MyMalJPEG features

applied on any file format. Generic feature extraction methods work on the file's building blocks (byte or character representation) in order to extract features that represent the file. In their research, they used three methods: a **simple histogram**, an advanced byte **entropy histogram** and **Min-Hash**.

It was decided to see whether manipulation of the image would impact the final answer. A simple histogram based on byte values (256 options) was used in our experiment. We used **equalization** and **quantization** as possible manipulations. Due to the *MalJPEG* paper's demonstration of the Random Forest classifier on all datasets created using histogram methods, we only present its detection results (Figure 9).

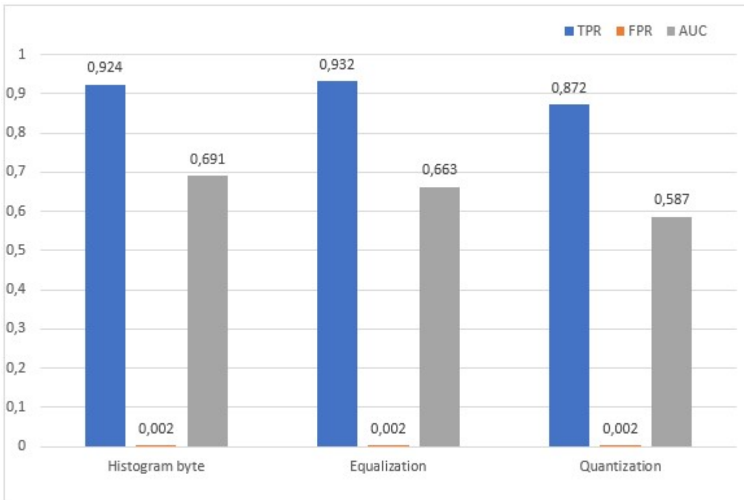

Figure 9: Detection results for the Random Forest classifier

5 Discussion and Conclusion

In this paper, we present *MyMalJPEG*, a machine learning-based solution for efficient detection of unknown malicious JPEG images. *MyMalJPEG* extracts 27 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.

In the first experiment, we compared the detection results of machine learning classifiers evaluated on datasets created using *MyMalJPEG* features, against *MalJPEG* features. As a result, *MyMalJPEG* performed better when using the *LightGBM*, *XGBoost*, and *Random Forest* classifiers: $TPR = 0.993$, $FPR = 0$, and $AUC = 0.996$. However, the results of this experiment also showed that the LightGBM classifier trained on MalJPEG features provides worse results: TPR = 0.951, FPR = 0.004, and AUC = 0.997[1].

In the second experiment, we examined how manipulation of images affects the results. Images that are benign are not particularly affected by **quantization** or **equalization**. However, a significant number of malicious images have been identified as benign.

Given the threats posed against individuals, businesses, and organizations by cyber attackers using malicious JPEG images, a comprehensive detection method is clearly required. *MyMalJPEG* provides efficient detection of known and unknown malicious JPEG images. *MyMalJPEG* works relatively fast, thus supporting real-time performance requirements for the detection of large image streams.

References

[1] Aviad Cohen Nir Nissim and Yuval Elovici. “MalJPEG: Machine Learning Based Solution for the Detection of Malicious JPEG Images”. In: *IEEEAccess* (2020), pp. 19997–20011.