Detection of malicious JPG Images Using Machine Learning Methods

Liad Nahmias, Alex Lerman, Aviad Cohen, Nir Nissim, Yuval Elovici

Department of Information Systems Engineering, Ben-Gurion University of the Negev, Israel

Malware Lab, Cyber Security Research Center, Ben-Gurion University of the Negev, Beer-Sheva, Israel

TABLE OF CONTECTS:

I. ABSTRACT

In the last decade, we are witnessing an increasing in Cyber-attacks, hacking actions and exploitation of vulnerabilities. Hackers, espionage agencies and even countries use cyber-attacks in order to earn money, get private information and make damage to the victim's equipment. A detection of those Cyber-attacks could help the user to prevent leaking of private and important data.

One of the most common ways to launch a cyber-attack is using malicious files. Non-executable files considered safe by most of the users and enables the attacker to perform malicious actions in practice.

The existing detection methods e.g. Anti-Virus softwares, FireWalls etc. are not good enough for early detetcion of maicious files and zero-days attacks, especially, in Images.

In our Project we examined and demonstrated two main feature selection methods, using machine learning techniques, for early detection of malicious RTF and JPG files. The first one is based on the structure of the specific file while the second methods is based on learning in depth the structure of the general files structure, former attacks and vulnerabilities.

We conducted and compared the detection ratio of those methods in three different experiments. we showed that both of the methods are a good way for early detection that surpass known Anti-Virus softwares.

Keywords — JPG, JPEG, Detection, Machine Learning, Analysis, Feature Extraction, knowledge-based, file structure

II. INTRODUCTION

In the last decade, we are witnessing an increasing in Cyber-attacks, hacking actions and exploitation of vulnerabilities. Hackers, espionage agencies and even countries use cyber-attacks in order to earn money, get private information and make damage to the victim's equipment.

One of the most common ways to launch a cyber-attack is using malicious files. Since executable files considered dangerous, attackers tend to leverage non-executable files in order to launch cyber-attacks. Non-executable files considered safe by most of the users and enables the attacker to perform malicious actions in practice.

Nowadays, Images are a very common file to share information and knowledge. Social media, advertising, marketing and more fileds using images expressing their feeling, share moments, saling items and even for educational usage. There are a lot of types of images e.g. JPG, GIF,BMP, TIFF etc.

Our research will be focusing on **Joint Photographic Experts Group** (JPEG) images, known as the most used among the image formats in the web[1]. JPEG has been an international standard for compressing digital still images since 1992. JPEG is a lossy compression algorithm (reserved only the necessary data) what makes it light to transfer through the web.

Locky ransomware spreads via Images on Facebook[2] **-** On November 25, 2016 images were sent through the facebook to many users. The image was SVG type and was a downloader of Locky Ransomware. The author emphasised that even JPEG files can be used as downloaders of Locky.

Detection of malicious JPEG images has great significant since JPEG images are used massively in today's website, emails etc. Attackers that exploit JPEG images can damage a lot of users, companies and systems.

In this research we are using static analysis (examining the file without running) in conjunction with machine learning (ML) in order to detect malicious JPEG images. We focus on the feature extraction process which aimed at extracting discriminative features from the examined documents in order to improve the detection capabilities of machine learning algorithms. In this research we propose two different approaches for feature extraction aimed at JPEG images. The first approach proposes a set of knowledge-based features. The second approach proposes a new feature extraction methodology that leverages the structure of the document and transform it to features. We extensively evaluate these two approches and compare them in a set of experiments.

The rest of the document is organizes as follows. We provide background information related to our research in Section 0III. Section EIV presents related work. In Section **Error! Reference source not**

[1] http://1stwebdesigner.com/image-file-types/
[2] http://thehackernews.com/2016/11/facebook-locky-ransomware.html

found. we describe the methods used in this research. Section C contains our evaluation and results. We discuss the results and security aspects, and present our conclusions, limitations, and future work in Section 1.

III. Background

JPEG is an abbreviation for Joint Photographic Experts Group. JPEG known as the most popular image format in the web. In 1992, JPEG became an international standard for compressing digital still images. It has two main internal formats:

1. **EFIF -** most common format for storing and transmitting photographic images on the World Wide Web.
2. **EXIF** - most common image format used by digital cameras and other photographic image capture devices.

As a compression algorithm, we can divide the JPEG process to four main levels[3]:

1. **Preprocessing -** Divide the image to 8X8 pixels blocks, every block called Minimum Coded Units (MCU). If the picture is colorful, change RGB to YCbCr (For math usage. The YCbCr is family of color space used as a part of the color image pipeline in digital photography systems[4]). Turn every block to matrix of the intensity of the pixels inside the block and subtract 127 (to median the value to 0).
2. **Transformation** - using Discrete Cosine Transform (DCT). The basic purpose of this operation is to take a signal and transform it from one type of representation to another. Generally, the DCT creates new image by the formula $C = U \cdot B \cdot U^T$ (T is transposed matrix), when B is the 8X8 block and U is a special matrix. By using DCT, the high intense parts of the block move to the left-upper side.
3. **Quantization -** round the near zero values to zero and the other to integer values. Thus, the Huffman coding algorithm will be more effective.
4. **Encoding** – JPEG use Huffman coding for this stage. Sometimes, JPEG can use an arithmetic-coding instead of Huffman.

JPEG is represented as compressed binary file and has three ways to show the compressed data:

1. **Interchange format for compressed image data -** includes all the tables that are required by the decoder.
2. **Abbreviated format for compressed image data** - may omit some or all of the tables needed for decoder. (The application must inherited those tables from a previous compressed data stream or must knew them through other mechanism).

[3] http://www.whydomath.org/node/wavlets/basicjpg.html
[4] https://en.wikipedia.org/wiki/YCbCr

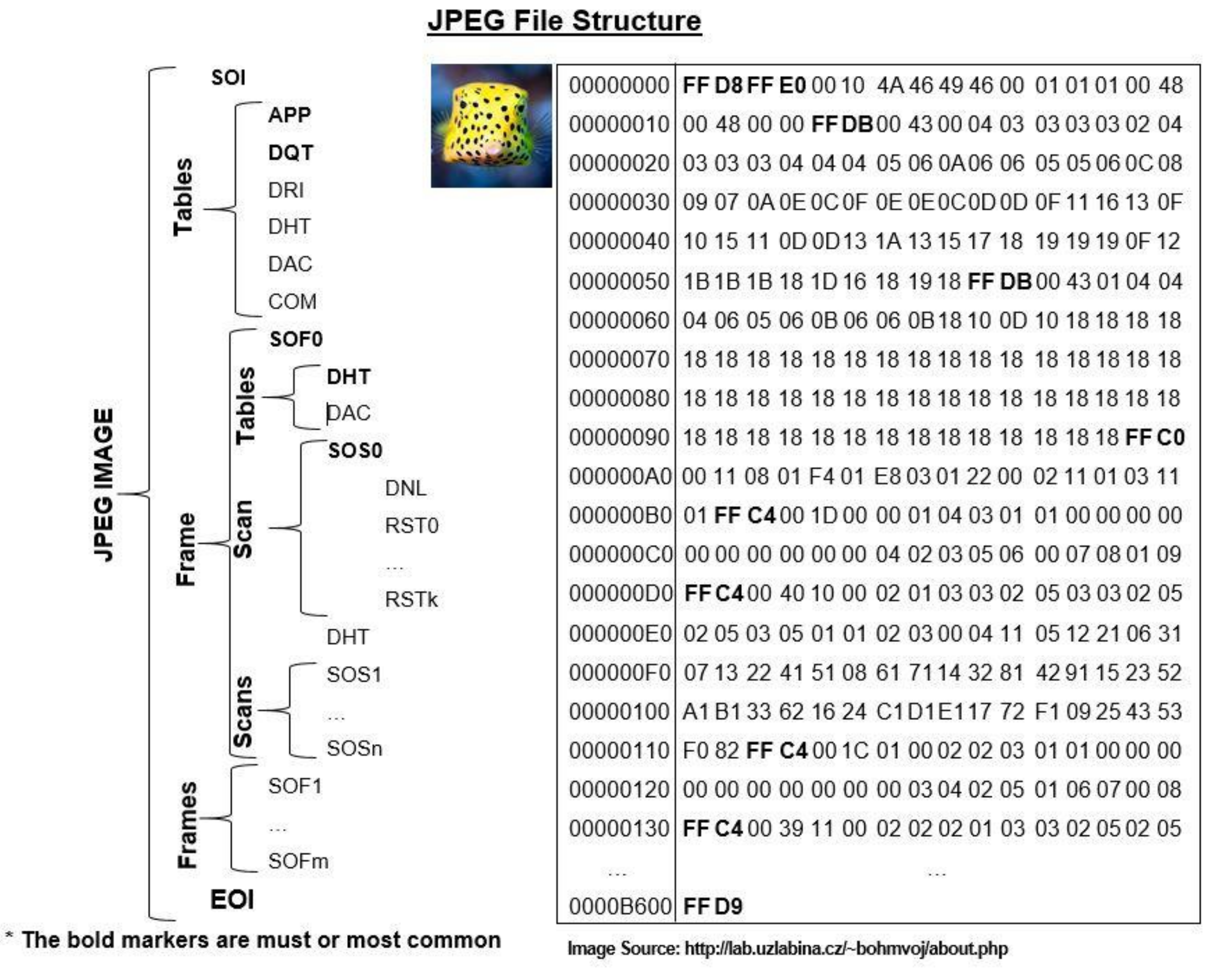
3. **Abbreviated format for table specification data** - contains no frame, scan or entropy-coded segments. If this format is in used, the application is responsible for coordinating table specification and compressed image data.

There are four main modes of JPEG:

1. **Baseline** - lossy, not progressive, use Huffman coding. Every application that supports JPEG, at least, supposed supporting Baseline JPEG. Identified by presence of marker *SOF0*.
2. **Progressive** - Rearrange the image data, so that the first part of it represents a very low quality version of the entire image, rather than a high-quality version of a small part of the image. (Identified by presence of markers: *SOF2, SOF6, SOF10 or SOF14*.
3. **Lossless** – JPEG supports true lossless compression. Identified by presence of markers: *SOF3, SOF7, SOF11 or SOF15*.
4. **Hierarchal (Differential)** - Similar to progressive JPEG but geared toward storing multiple sizes of the same image, such that the decoder can select the size it prefers. Identified by presence of markers: *SOF5, SOF6, SOF7, SOF13, SOF14 or SOF15*.

A. *JPEG File Structure*

JPEG file, like other digital formats, is a Binary file and its data can represent by Hexadecimal view mode. Figure 1 presents the structure of JPEG file. In the right side, we can see the hierarchy of the markers and the division to frames and scans (see explanation below the figure). In the left side, we can see the hexadecimal of an image.



```
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 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 (hierarchy and hex view).

JPEG file uses two main classes of segments: **Marker Segments** and **Entropy-coded segments**.

1. **Marker segments -** contains header information, tables (quantization tables, entropy-coding tables etc.) and other information required to interpret and decode the compressed image data. The markers help to divide the file to different segments. Every marker starts with 0xFF and it length is 2 bytes.

Marker Name	Marker Hex Code	Definition
APP$_n$	0xFFE0 - 0xFFEF	Reserved for application used
COM	0xFFFE	Comment

DAC	0xFFCC	Define arithmetic conditioning table(s)
DHP	0xFFDE	Define hierarchical progression
DHT	0xFFC4	Define Huffman table(s)
DNL	0xFFDC	Define number of lines
DQT	0xFFDB	Define quantization table(s)
DRI	0xFFDD	Define restart interval
EOI	0xFFD9	End of image
EXP	0xFFDF	Expand reference image(s)
JPG	0xFFC8	Reserved for JPEG extensions
JPG$_n$	0xFFF0- 0xFFFD	Reserved for JPEG extensions
RES	0xFF02- 0xFFBF	Reserved
RST$_m$	0xFFD0- 0xFFD7	Restart with modulo 8 counter m
SOF$_n$	0xFFC0-3, 5-7, 9-B, D-F	Start of frame
SOI	0xFFD8	Start of image
SOS	0xFFDA	Start of scan
TEM	0xFF01	For temporary use in arithmetic coding

2. *Table 1* presents the possible markers, their Hex code and the definition of their role. There are some special cases we are need to be aware with when we are observing JPEG's markers:
   2.1. XFF00 is not a marker but part of the image data.
   2.2. All the markers except of FFD0 to FFD9 and FF01 (reserved), is immediately followed by a length specifier (gives the length of that marker segment, exclude the marker).
   2.3. The DHT, DAC, DRI, DQT, COM and APP$_n$ marker segments may be inserted into the compressed data before the frame and scan headers. They may be used in any order and as many times as desired.
   2.4. If the DHT and DQT marker segments follow the start-of-image marker (SOI), they may be followed immediately by the end-of-image (EOI) marker. This is the JPEG abbreviated format for table specification data. COM and APP$_n$ marker segments may also be used in this abbreviated sequence.
   2.5. The DAC and DRI may appear but have no function, because the next SOI will set default values.

2.6. The marker terminating the final restart interval of the first scan may be preceded by the DNL marker.
2.7. The last entropy-coded segment in a scan is not followed by an RST_m marker.
2.8. Image data stream contains a single image that can have up to 255 unique components.

Marker Name	Marker Hex Code	Definition
APP_n	0xFFE0 - 0xFFEF	Reserved for application used
COM	0xFFFE	Comment
DAC	0xFFCC	Define arithmetic conditioning table(s)
DHP	0xFFDE	Define hierarchical progression
DHT	0xFFC4	Define Huffman table(s)
DNL	0xFFDC	Define number of lines
DQT	0xFFDB	Define quantization table(s)
DRI	0xFFDD	Define restart interval
EOI	0xFFD9	End of image
EXP	0xFFDF	Expand reference image(s)
JPG	0xFFC8	Reserved for JPEG extensions
JPG_n	0xFFF0- 0xFFFD	Reserved for JPEG extensions
RES	0xFF02- 0xFFBF	Reserved
RST_m	0xFFD0- 0xFFD7	Restart with modulo 8 counter m
SOF_n	0xFFC0-3, 5-7, 9-B, D-F	Start of frame
SOI	0xFFD8	Start of image
SOS	0xFFDA	Start of scan
TEM	0xFF01	For temporary use in arithmetic coding

Table 1. List of possible JPEG's markers.

Elaborate of Markers Definition:

1.1. **APP_n -** The APP_n (Application) marker segments are reserved for application use. Since these segments may be defined differently for different applications, the JPEG DIS states that they should be removed when the data are transferred between application environments. However, the reader should be aware that JPEG did not make the removal of these APP$_n$ marker segments mandatory. Therefore, if two applications require conflicting usage of an APP$_n$ marker segment, checking of the data may be needed when JPEG compressed data is imported or exported from one application to the other. Table 2 presents the parameters of APP marker - their name, parameter symbol and the size of it in bits. Table 3 presents a list of possible applications and their APP marker. There are some special cases we are need to be aware with when we are observing APP marker:

1.1.1. Major Revision number should be valued as 1 – otherwise error.

1.1.2. Minor Revision number should be valued 0, 1 or 2.

1.1.3. Units densities – 0x00= no units (x/y specified as ratio), 0x01=pixel per inch and 0x02 = pixel per centimeter.

1.1.4. X-density and Y-density supposed to be not zero value.

Parameter Name	Parameter Symbol	Parameter Size (bits)
Marker (0xFFE0-0xFFEF)	APP$_n$	16
APP definition length	Lp	16
File identifier mark	AP$_i$	40
Major Revision number	?	8
Minor Revision number	?	8
Units for x/y densities	?	8
X-density	?	16
Y-density	?	16
Thumbnail width	?	8
Thumbnail height	?	8
Bytes to be read	?	3*width*height

Table 2. APP marker's Parameters

Segment	Marker Hex Code
JFIF	APP0 (0xFFE0)
JFXX	APP0 (0xFFE0)
EXIF	APP1 (0xFFE1)

XMP	APP1 (0xFFE1)
ICC	APP2 (0xFFE2)
META	APP3 (0xFFE3)
Picture Info	APP12 (0xFFEC)
Ducky	APP12 (0xFFEC)
Photoshop IRB	APP13 (0xFFED)
Adobe	APP14 (0xFFEE)

Table 3. List of possible application and their specific APP marker[5]

1.2. **COM** - The COM (Comment) marker segment is intended for text fields. It may not contain information that could affect decoding. JPEG has not specified the character set to be used in this segment.

1.3. **DAC** - The DAC (Define Arithmetic Coding Conditioning) marker segment defines one or more tables of parameters for the conditioning of the probability estimates used in the arithmetic coding procedures. Table 4 presents the parameters of DAC marker - their name, parameter symbol and the size of it in bits. There are some special cases we are need to be aware with when we are observing DAC marker:

1.3.1. Tc is 0 for DC and lossless coding, 1 for AC tables.

1.3.2. Ta can be valued as 0, 1, 2 or 3.

1.3.3. Cs is built by two parameters L and U valued from 0 to 15. These two parameters control the conditioning of some of the probabilities used in coding the DPCM differences.

Parameter Name	Parameter Symbol	Parameter Size (bits)
Marker (0xFFCC)	DAC	16
Arithmetic coding conditioning table length	La	16
For each arithmetic-coding table:		
Table class	Tc	4
Quantization table identifier	Ta	4
Conditioning table value	Cs	8

Table 4. DAC marker's Parameters

[5] http://www.ozhiker.com/electronics/pjmt/jpeg_info/app_segments.html

1.4. **DHP** - The DHP (Define Hierarchical Progression) marker segment is used to signal the parameters identifying the dimensions, components and relative sampling of the final image of the hierarchical progression. It uses the same syntax as the frame header and must precede the first frame in the progression.

1.5. **DHT -** The DHT (Define Huffman Table) marker segment defines one or more Huffman tables.

1.6. Table 5 presents the parameters of DHT marker - their name, parameter symbol and the size of it in bits. There are some special cases we are need to be aware with when we are observing DHT marker:

1.6.1. Tc 0 for DC and lossless coding, 1 for AC tables.

1.6.2. Th can be valued as 0,1,2 or 3 except in the baseline DCT system, in which the value is restricted to 0 and 1.

1.6.3. Vij maximum value is 255.

Parameter Name	Parameter Symbol	Parameter Size (bits)
Marker (0xFFC4)	DHT	16
Huffman table definition length	Lh	16
For each Huffman table:		
Table class (AC or DC)	Tc	4
Huffman table identifier	Th	4
Number of Huffman codes of length i (for i=1 to 16)	Li	8
Value associated with each Huffman code (for i=1 to 16; j=1 to Li)	Vij	8

Table 5. DHT marker's Parameters

1.7. **DNL** – The DNL (Define Number of Lines) marker segment provides a mechanism for defining or redefining the number of lines in the image at the end of the first scan. The two bytes length (always 4) is followed by a two byte value giving the number of lines in the frame. The number of lines specified in the DNL marker segment must be consistent with the number of lines and the sampling factors already specified in the frame header and with the number of lines already decoded. Thus, unless the value specified in the frame header is zero, the number of lines specified in the DNL marker segment must be less than or equal to the frame header value, and in all cases must be a value somewhere within the range of the last row of MCU decoded. Note that if the data are directly written to an image buffer, spurious data may be written to the buffer if rows of samples are padded in this last MCU row.

1.8. **DQT -** The DQT (Define Quantization Table) marker segment defines one or more quantization tables. Quantization tables used in a progression should not be changed between scans of that progression. The DQT marker has meaning only for DCT-based algorithm. Table 6 presents the parameters of DQT marker - their name, parameter symbol and the size of it in bits. There are some special cases we are need to be aware with when we are observing DQT marker:

1.8.1. Pq – 0 for 8-bit precision and 1 for 1 16-bit precision**.**

1.8.2. Tq can be valued as 0, 1, 2 or 3.

1.8.3. Qk may have values from 1 to the maximum value permitted for the precision specified.

Parameter Name	Parameter Symbol	Parameter Size (bits)
Marker (0xFFDE)	DQT	16
Quantization table definition length	Lq	16
For each quantization table:		
Quantization table element precision	Pq	4
Quantization table identifier	Tq	4
Quantization table element (k=0,…, 63)	Qk	8 or 16

Table 6. DQT marker's Parameters

1.9. **DRI -** The DRI (Define Restart Interval) marker segment provides the mechanism for setting and resetting the restart interval. The two-byte length (always 4) is followed by a two-byte value giving the number of MCU in the restart interval. The restart interval is set to zero at the start of an image. When the restart interval is zero, the restart function is disabled. Table 7 presents the parameters of DRI marker - their name, parameter symbol and the size of it in bits. Each Restart interval in scan, except the final one - contains only the number of MCU required to complete the scan):

Parameter Name	Parameter Symbol	Parameter Size (bits)
Marker (0xFFDD)	DRI	16
Define restart interval segment length	Lr	16
Restart interval	Ri	16

Table 7. DRI marker's Parameters

1.10. **EOI -** The EOI (End of Image) marker terminates the JPEG compressed data stream.

1.11. **EXP -** The EXP (Expand) marker segment signals the expansion of the spatial resolution of the reference data needed by the hierarchical differential frame that follows. It is followed by a two byte length (always 3) and an eight bit integer that has one of the values 0x10, 0x01 or 0x11. 0x10 signals a horizontal expansion by a factor of 2, 0x01 signals a vertical expansion by a factor 0f 2, and 0x11 signals both horizontal and vertical expansion by a factor of 2.

1.12.**JPG, JPG$_n$ -** The JPG and JPG$_n$ marker segments are reserved for future JPEG extensions.

1.13.**RES -** The RES markers are reserved for future JPEG extensions.

1.14.**RTS$_m$ -** The RST$_m$ (Restart) marker is appended to the compressed data between restart intervals. This provides a unique byte-aligned code that can be located by scanning the compressed data. A three bit field (m) in this marker code provides a modulo-8 restart interval count. The modulo count of the first RST$_m$ code in a scan is zero.

1.15.**SOF$_n$ -** An SOF$_n$ (start of Frame) marker begins a frame header. The variable length SOF$_n$ marker segment gives the frame parameters that apply to all scans within the frame. The particular value of n in SOF$_n$ identifies the mode of compression and the entropy coder used within the frame. Table 8 presents a list of possible frames and their SOF marker.

Segment	Marker Hex Code	Definition
SOF0	0xFFC0	Nondifferential Huffman coding frame, Baseline DCT
SOF1	0xFFC1	Nondifferential Huffman coding frame, Extended sequential DCT
SOF2	0xFFC2	Nondifferential Huffman coding frame, Progressive DCT
SOF3	0xFFC3	Nondifferential Huffman coding frame, Lossless (sequential)
SOF5	0xFFC5	Differential Huffman coding frame, Differential sequential DCT
SOF6	0xFFC6	Differential Huffman coding frame, Differential progressive DCT
SOF7	0xFFC7	Differential Huffman coding frame, Differential lossless
SOF9	0xFFC9	Nondifferential arithmetic coding frame, Extended sequential DCT
SOF10	0xFFCA	Nondifferential arithmetic coding frame, Progressive DCT
SOF11	0xFFCB	Nondifferential arithmetic coding frame, Lossless (sequential)
SOF13	0xFFCD	Differential arithmetic coding frame, Differential sequential DCT
SOF14	0xFFCE	Differential arithmetic coding frame, Differential progressive DCT
SOF15	0xFFCF	Differential arithmetic coding frame, Differential lossless

Table 8. List of possible frames and their specific SOF marker

1.16.**SOI -** The SOI (Start of Image) marker begins the compressed data stream. Note that the SOI marker can be used to detect problems with bit order, bit sense and bit alignment of the data.

1.17. **SOS -** The SOS (Start of Scan) marker begins a scan header. The scan header is always followed immediately by entropy-coded data for the scan. The two-byte length gives the number of bytes of scan parameters.

1.18. **TEM -** The TEM marker is reserved for temporary private use by arithmetic encoders. It is used to signal temporarily an unresolved carry-over, so that post processing can carry out the resolution offline. It may not occur in JPEG compressed data streams.

3. **Entropy-coded segments -** contains the entropy-coded data (comes after start of scan [SOS]).

A JPEG compression process consists of a single **Frame** or sequence of frames. Each Frame consists on or more **Scan** and Scan is a single pass through the data for one or more components of the image.

1. **Frame**:

   Table 9 presents the parameters of a frame - their name, parameter symbol and the size of it in bits. We must be aware to some identifiers:

   1.1. Before every frame, there is a header contains the needed values for decoding.

   1.2. For non-hierarchical processes, the frame defines the basic attributes of the image, including size, number of components, precision and entropy-coding technique.

   1.3. For non-hierarchical encoding, only a single frame is allowed, whereas in the hierarchical mode a sequence of frames may be use.

   1.4. All the integer parameters in the frame header are unsigned.

   1.5. The parameters following the Start-Of-Frame marker are:

   1.5.1. **Frame header length -** A 16-bit integer, **Lf**, gives the length in bytes of the frame parameters. Exclude the SOF marker bytes but include the two bytes in the length field.

   1.5.2. **Sample Precision -** An eight-bit integer, **P**, specifies the sample precision in bits. A precision of either eight or twelve bits is allowed for the DCT compression algorithm and precisions from two to sixteen bits are allowed for the lossless DPCM coding algorithms. All image components in a frame must have the same sample precision.

   1.5.3. **Number of lines -** A 16-bits integer, **Y**, specifies the number of raster lines in the internal representation of the frame. The value excludes any lines added to complete the bottom row of MCU in the frame. If the number of lines is set to zero, it is unspecified at the start of compression. In this case, the DNL segment must be at the end of the first scan of the frame to signal the number of lines.

   1.5.4. **Number of samples per line -** A 16-bits integer, **X**, specifies the number of samples per raster line in the internal representation of the frame. The value excludes any columns added to complete the MCU at the right edge of the frame.

   1.5.5. **Number of components in frame -** An eight-bit integer, **Nf**, specifies the number of components in the frame. Nf can have any value from 1 to 255 in sequential DCT-based frames and lossless frames, In progressive frames, Nf may have only values from 1 to 4. A value of zero is undefined and is forbidden.

1.5.6. **Frame component specification parameters -** Each component in a frame consists a component identifier (**C_i**), the horizontal (**H_i**) and vertical (**V_i**) sampling factors and the quantization table selector (**Tq_i**) must be specified. For Component identifier, any eight-bit value is allowed and each component must be assigned a unique number that is used for that component in the scans in the image. The sampling factors pair (H_i,V_i) makes up an eight-bit integer (first 4 bits are the horizontal and the last 4 bits are the vertical). The allowed values of H_i and V_i are 1, 2, 3 and 4, all other values being forbidden. The Tq_i parameter may have values of 0, 1, 2 or 3 (defines the id of the quantization table – 1 is the quantization number 1 etc.). It selects the quantization table to be used for the ith component. A maximum of four quantization tables may be defined.

Parameter Name	Parameter Symbol	Parameter Size (bits)
Marker (0xFFC0-3, 5-7, 9-B, D-F)	SOF_n	16
Frame header length	Lf	16
Sample Precision	P	8
Number of lines	Y	16
Number of samples per line	X	16
Number of components in frame	Nf	8
Frame component specification parameters:		
component identifier	C_i	8
Horizontal sampling factor	H_i	4
Vertical sampling factor	V_i	4
Quantization table destination selector	Tq_i	8

Table 9. Frame's Parameters

2. **Scan:**

There are some Table 10 presents the parameters of a scan - their name, parameter symbol and the size of it in bits. We must be aware to some identifiers:

2.1. Before every scan, there is a header contains the needed values for decoding.
2.2. Every scan header is always followed by one or more entropy-coded segments.

2.3. The input data to a scan may be divided into fixed intervals called "restart intervals". The data in each Restart interval are compressed into a single entropy-coded segment.

2.4. The number of components in the scan dictates the type of data ordering used in the scan. If the has only one component, the data are non-intervals and each MCU contains one data unit. If the scan contains more than one component, the data are interleaved and the number of data units in the MCU is determined by the sampling factors of the component in the scan.

2.5. MCU – If more than one component is specified for the scan, the components in the scan and their sampling factors (specified in the frame header) define the ordering of components and the number of data units (blocks or samples, depending on the mode) of each component in the MCU. The total number of data units in the MCU, Nb is defined: $Nb = \sum_{s=1}^{Ns} H_{i(s)} * V_{i(s)}$

The total number of data units in the MCU may not exceed 10. Therefore, any combinations of components and sampling factor that would give Nb>10 are forbidden.

2.6. The parameters following the Start-Of-Scan marker are:

2.6.1. **Scan header length -** A 16-bit integer, **Ls**, gives the length in bytes of the scan header. Exclude the SOF marker bytes but include the two bytes in the length field.

2.6.2. **Number of components in scan -** An eight-bit integer, **Ns**, specifies the number of components in a scan. Ns may have only values from 1 to 4.

2.6.3. **Scan component specification parameters -** The components selected for a scan may be a subset of the components specified for the frame, cut must follow the same order as in the frame component specification. Each component in a scan, a scan component selector and entropy-coding table selectors must be specified and consists a scan component selector for the kth scan component (**Cs_k**), DC entropy-coding table selector for the kth scan component (**Td_k**) and AC entropy-coding table selector for the kth scan component (**Ta_k**). The identification numbers (Cs_1,…, Cs_{Ns}) must match one of the component identification numbers specified in the component specification defined for the frame. The entropy-coding tables pair (Td_k, Ta_k) makes up an eight-bit integer (first 4 bits are the Td_k and the last 4 bits are the Ta_k). In extended systems, the allowed values of Td_k and Ta_k are 0, 1, 2 or 3, whereas for the baseline system the allowed values are 0 and 1.

2.6.4. **Start of spectral selection or predictor selection -** The start of spectral selection, Ss, specifies the starting index of a band of coefficients (in zigzag or contiguous order) is coded in the scan. For a sequential process Ss is zero, whereas for a progressive DCT scan, Ss may have any value from 0 to 63.

2.6.5. **End of spectral selection -** The end of spectral selection, Se, specifies the index of the last coefficient in the spectral band. For a progressive DCT scan, Se may have any value from Ss to 63, whereas for a DCT sequential process the end of spectral selection, Se, is set to 63. If Ss is 0, Se must be 0 too.

2.6.6. **Successive approximation bit position high -** The successive approximation bit position high parameter, Ah, is set to the value of Al of the preceding scan of the same band. If there was no preceding scan of the same band, Ah is set to zero. Ah set to zero also for lossless mode. If successive approximation is used in the first scan of a spectral-selection band, the coefficients are coded at reduced precision.

2.6.7. **Successive approximation bit position low or point transform –** A four-bit integer, Al, specifies the point transform used to reduce the precision of the DCT coefficients. The AC coefficients in the band are scaled by dividing them by 21^{Al}, whereas the DC coefficients are scaled by an arithmetic-right-shift by Al. Effectively, Al gives the magnitude least significant bit position after scaling. For every scan except the first one, the Al must be one unit smaller than Ah. At the first scan, if Ah is zero the Al set to the desire point transform value.

Parameter Name	Parameter Symbol	Parameter Size (bits)
Marker (0xFFDA)	SOS	16
Scan header length	Ls	16
Number of components in scan	Ns	8
Scan component specification (k=1,…, Ns)		
Scan component selector	Cs_k	8
DC entropy-coding table selector	Td_k	4
AC entropy-coding table selector	Ta_k	4
Start of spectral selection or predictor selection	Ss	8
End of spectral selection	Se	8
Successive approximation bit position high	Ah	4
Successive approximation bit position low or point transform	Al	4

Table 10. Scan's Parameters

B. Attack Techniques

As we mentioned in the introduction, we are witness growth in hacking action all over the globe. Everyday hackers find new techniques or new ways to improve their former version of codes to succeed infecting more machines. It is just matter of time, hackers will used more intensively JPEG format, known as the most used among the image formats in the web, to fulfil their goals. For now, we can put our finger on some of the trends that can be used for hacking, involving JPEG files:

1. **Steganography**[6] (Steganos – covered, graphie - "writing") - A technique of disguise information inside file without damage it. Hackers can use this technique for hiding their code inside the JPEG file. Thus, the malicious code is hiding (e.g in the pixels) from the eye of the user and it hard to impossible for detecting.
2. **Vulnerabilities of the viewer software –** Displaying JPEG file on the computer is required viewer software. Many of JPEG viewers have vulnerabilities by themself. Hackers can exploit those vulnerabilities and create, for example buffer over-flow or heap-spray. With this technique, the hacker can gain many of the user permissions.
3. Downloader[7] - One of the familiar ways of hacking to a computer it is by sending a file that contains a URL for downloading the malicious code. It can be .doc, .jpg, .pdf etc. on November 25, 2016, hackers used image (.svg) to download a "Locky" executable. Another known way is using the file structure to add the URL as comment (FF FE).

[6] http://thehackernews.com/2015/06/Stegosploit-malware.html, https://blog.sucuri.net/2013/07/malware-hidden-inside-jpg-exif-headers.html, http://php.webtutor.pl/wp-content/uploads/2011/04/php-logo-virus.jpg, http://www.idigitaltimes.com/hacking-pictures-new-stegosploit-tool-hides-malware-inside-internet-images-instant-444768, http://www.pcworld.com/article/2105408/3/watch-out-for-photos-containing-malware.html

[7] http://thehackernews.com/2016/11/facebook-locky-ransomware.html, http://securityaffairs.co/wordpress/53650/malware/svg-images-locky.html, https://blog.malwarebytes.com/threat-analysis/2014/02/hiding-in-plain-sight-a-story-about-a-sneaky-banking-trojan/, http://blog.trendmicro.com/trendlabs-security-intelligence/jpeg-files-used-for-targeted-attack-malware/

C. *List of CVEs*

Vulnerability is a weakness in an application that arises from the way the application is written. A vulnerability in an application could allow an attacker to compromise the integrity, availability, or confidentiality of the application. CVE[8] (Common Vulnerabilities and Exposures) is a standard for information security vulnerability names. The MITRE Corporation maintains the CVE system which is a free, publicly available dictionary of known information security vulnerability and exposures, subsidized by the National Cyber Security Division of United States Department of Homeland Security. The CVE-ID format is: "CVW-YYYY-NNNN" (YYYY – 4-digit Year, NNNN – 4-digit number). Table 11 presents a list of known CVEs for JPEG images.

ID	CVE Name	Description	Effect
1	CVE-2016-5159	openjpeg: A heap-based buffer overflow flaw was found in the patch for CVE-2013-6045. A crafted j2k image could cause the application to crash, or potentially execute arbitrary code.[9]	Allows remote attackers to cause a denial of service (heap-based buffer overflow) or possibly have unspecified other impact.[10]
2	CVE-2016-5158	Multiple integer overflows in the opj_tcd_init_tile function in tcd.c in OpenJPEG, as used in PDFium in Google Chrome before 53.0.2785.89 on Windows and OS X and before 53.0.2785.92 on Linux, allow remote attackers to cause a denial of service (heap-based buffer overflow) or possibly have unspecified other impact via crafted JPEG 2000 data.[11]	Allows remote attackers to cause a denial of service (heap-based buffer overflow) or possibly have unspecified other impact via crafted JPEG 2000 data.[12]

[8] http://cve.mitre.org/
[9] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5159
[10] https://www.rapid7.com/db/vulnerabilities/redhat_linux-cve-2016-5159
[11] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5158
[12] https://www.rapid7.com/db/vulnerabilities/google-chrome-cve-2016-5158

3	CVE-2016-5152	Integer overflow in the opj_tcd_get_decoded_tile_size function in tcd.c in OpenJPEG, as used in PDFium in Google Chrome before 53.0.2785.89 on Windows and OS X and before 53.0.2785.92 on Linux, allows remote attackers to cause a denial of service (heap-based buffer overflow) or possibly have unspecified other impact via crafted JPEG 2000 data.[13]	Allows remote attackers to cause a denial of service (heap-based buffer overflow) or possibly have unspecified other impact via crafted JPEG 2000 data.[14]
4	CVE-2016-5140	Heap-based buffer overflow in the opj_j2k_read_SQcd_SQcc function in j2k.c in OpenJPEG, as used in PDFium in Google Chrome before 52.0.2743.116, allows remote attackers to cause a denial of service or possibly have unspecified other impact via crafted JPEG 2000 data.[15]	Allows remote attackers to cause a denial of service or possibly have unspecified other impact via crafted JPEG 2000 data.[16]
5	CVE-2016-5139	Multiple integer overflows in the opj_tcd_init_tile function in tcd.c in OpenJPEG, as used in PDFium in Google Chrome before 52.0.2743.116, allow remote attackers to cause a denial of service (heap-based buffer overflow) or possibly have unspecified other impact via crafted JPEG 2000 data.[17]	Allows remote attackers to cause a denial of service (heap-based buffer overflow) or possibly have unspecified other impact via crafted JPEG 2000 data.[18]

[13] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5152
[14] https://www.rapid7.com/db/vulnerabilities/google-chrome-cve-2016-5152
[15] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5140
[16] https://cve.circl.lu/cve/CVE-2016-5140
[17] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5139
[18] https://cve.circl.lu/cve/CVE-2016-5139

6	CVE-2016-1924	The opj_tgt_reset function in OpenJpeg 2016.1.18 allows remote attackers to cause a denial of service (out-of-bounds read and application crash) via a crafted JPEG 2000 image.[19]	Allows remote attackers to cause a denial of service (out-of-bounds read and application crash) via a crafted JPEG 2000 image.[20]
7	CVE-2016-1923	Heap-based buffer overflow in the opj_j2k_update_image_data function in OpenJpeg 2016.1.18 allows remote attackers to cause a denial of service (out-of-bounds read and application crash) via a crafted JPEG 2000 image.[21]	Allows remote attackers to cause a denial of service (out-of-bounds read and application crash) via a crafted JPEG 2000 image.[22]
8	CVE-2016-1645	Multiple integer signedness errors in the opj_j2k_update_image_data function in j2k.c in OpenJPEG, as used in PDFium in Google Chrome before 49.0.2623.87, allow remote attackers to cause a denial of service (incorrect cast and out-of-bounds write) or possibly have unspecified other impact via crafted JPEG 2000 data.[23]	Allows remote attackers to cause a denial of service (incorrect cast and out-of-bounds write) or possibly have unspecified other impact via crafted JPEG 2000 data.[24]
9	CVE-2016-1628	pi.c in OpenJPEG, as used in PDFium in Google Chrome before 48.0.2564.109, does not validate a certain precision value, which allows remote attackers to execute arbitrary code or cause a denial of service (out-of-bounds read) via a crafted JPEG 2000 image in a PDF	Allows remote attackers to execute arbitrary code or cause a denial of service (out-of-bounds read) via a

[19] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1924
[20] https://cve.circl.lu/cve/CVE-2016-1924
[21] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1923
[22] https://cve.circl.lu/cve/CVE-2016-1923
[23] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1645
[24] https://cve.circl.lu/cve/CVE-2016-1645

		document, related to the opj_pi_next_rpcl, opj_pi_next_pcrl, and opj_pi_next_cprl functions.[25]	crafted JPEG 2000 image in a PDF document, related to the opj_pi_next_rpcl, opj_pi_next_pcrl, and opj_pi_next_cprl functions.[26]
10	CVE-2015-6776	The opj_dwt_decode_1* functions in dwt.c in OpenJPEG, as used in PDFium in Google Chrome before 47.0.2526.73, allow remote attackers to cause a denial of service (out-of-bounds array access) or possibly have unspecified other impact via crafted JPEG 2000 data that is mishandled during a discrete wavelet transform.[27]	Allows remote attackers to cause a denial of service (out-of-bounds array access) or possibly have unspecified other impact via crafted JPEG 2000 data that is mishandled during a discrete wavelet transform.[28]
11	CVE-2014-7903	Buffer overflow in OpenJPEG before r2911 in PDFium, as used in Google Chrome before 39.0.2171.65, allows remote attackers to cause a denial of service or possibly have unspecified other impact via a crafted JPEG image. [29]	Allows remote attackers to cause a denial of service or possibly have unspecified other impact via a crafted JPEG image.[30]

[25] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1628
[26] https://cve.circl.lu/cve/CVE-2016-1628
[27] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-6776
[28] https://cve.circl.lu/cve/CVE-2015-6776
[29] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-7903
[30] https://cve.circl.lu/cve/CVE-2014-7903

12	CVE-2012-3535	Heap-based buffer overflow in OpenJPEG 1.5.0 and earlier allows remote attackers to cause a denial of service (application crash) and possibly execute arbitrary code via a crafted JPEG2000 file. [31]	Allows remote attackers to cause a denial of service (application crash) and possibly execute arbitrary code via a crafted JPEG2000 file.[32]
13	CVE-2004-0200	Buffer overflow in the JPEG (JPG) parsing engine in the Microsoft Graphic Device Interface Plus (GDI+) component, GDIPlus.dll, allows remote attackers to execute arbitrary code via a JPEG image with a small JPEG COM field length that is normalized to a large integer length before a memory copy operation. [33]	An attacker who successfully exploited this vulnerability could take complete control of an affected system, including installing programs; viewing, changing, or deleting data; or creating new accounts with full privileges.[34]
14	CVE-2000-0655	Netscape Communicator 4.73 and earlier allows remote attackers to cause a denial of service or execute arbitrary commands via a JPEG image containing a comment with an illegal field length of 1. [35]	Allows remote attackers to cause a denial of service or execute arbitrary commands via a JPEG image containing a comment with an illegal field length of 1.[36]

Table 11. CVE list for JPEG files.

[31] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-3535
[32] https://cve.circl.lu/cve/CVE-2012-3535
[33] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0200
[34] https://www.rapid7.com/db/vulnerabilities/WINDOWS-HOTFIX-MS04-028
[35] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0655
[36] https://cve.circl.lu/cve/CVE-2004-0200

D. Analysis and Detection Tools

There are many detection tools available on the web. Every one of them detects malicious activities in different formats of files (only for Image, only office documents etc.) and used different ways (Static analysis/Dynamic analysis/both). For comparison usage, we summarized all the existing tools suggest detection for malicious JPEG files:

1. **Anti-virus softwares -** Generally, anti-virus softwares provides service of checking malicious files (Including JPEG) and provide a verdict.
2. **VirusTotal**[37] **-** A free web service that analyzes suspicious files, URLs and checking them with many Anti-virus softwares (more than 60). The service can detect viruses, worms, Trojans and all kind if malware and provides to the user a report with verdict.
3. **StegSecret**[38] **-** Steganalysis open source project that helps to detect hidden information in different digital media.
4. **VSL Tool**[39] **-** Steganalysis tool for detecting hidden code in JPEG files.

As we can see, after a thorough research for tools and services that detect malicious JPEG files, we barely found few. JPEG format is popular and is common used through the web but the options for detection malicious files are very narrow.

E. Approaches for Malware Detection

There are two fundamental methods for malware detection, the first is dynamic and the second is static. The main difference between those two methods is that in static analysis we don't run the code while in dynamic analysis we do run it.

Those two methods let you check in real-time the file and detect the malicious activities and code that can harsh your machine.

Both of them has cons and pros:

Static analysis will not run the malware so you can determinate the code on your own machine while using dynamic analysis will be required a virtual machine or other machine.

On the other hand, when you need to use static analysis on an obfuscated and bige size code, Dynamic analysis will help you just run it and see the code behavior, activities on the machine etc.

[37] http://www.virustotal.com/
[38] http://stegsecret.sourceforge.net/
[39] http://vsl.sourceforge.net/

Another known methods for malware detection is based on the file signature. File signature means to take the file and calculated his hash (SHA-1, SHA-256, MD5 etc.). In this way you create a unique identifier to the file. Anti-Virus softwares use this way and it works pretty well but this way has its own big disadvantage – if the Anti-Virus software has to give verdict on Zero-Day attack or even on a file it has in her Database but with a change of a character, this methods couldn't work well.

IV. RELATED WORK

In this section we present papers related to our research topic. The various articles and researches below helped us to choose our methodology of analysis (between static and dynamic) and the research work flow - Structural (aka feature extraction based on the file) and Knowledge-based (aka feature extraction based on former attacks and the file structure).

Ford, Cova, Kruegel and Kinda [FCKV00] showed the importance of both static and dynamic analysis when dealing with analysis and detection of malicious Flash advertisements. The authors created a tool calls "OdoSwiff" using static analysis to parse the tags of the flash. The parsing action is helping to detect known malicious techniques e.g. hiding malicious code (Shellcode or ActionScript code) and for finding a use of known vulnerabilities. "OdoSwiff" also uses dynamic analysis required to execute the Flash application and follows the execution trace in order to find out referenced URLs and network activities. For testing the success rate of "OdoSwiff", the authors collected corpuses of malicious Flash advertisements and compare the results to the detection rate of VirusTotal (a web platform that shows results of running files in many different Anti-Virus softwares). The results show that "OdoSwiff" got a higher detection rate, by 14% more than VirusTotal. Figure 2 presents those results of detection malicious flash advertisements out of 2,492 advertisements ("OdoSwiff" and "adopstools" produced one FP and "OdoSwiff" detected four malicious advertisements while "adopstools" detected three malicious advertisements.).

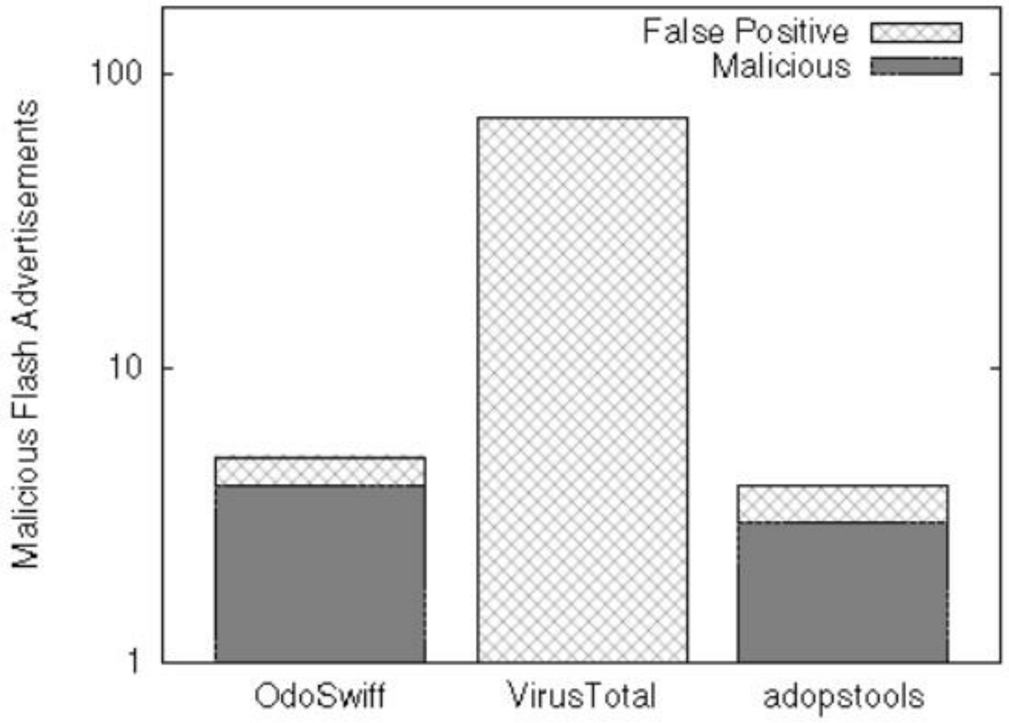


Figure 2. Detection results for crawled Flash advertisements

Article [JoKK00], deals with the vulnerabilities in web applications. The authors elaborated about the different vulnerabilities (SQL injection, cross-site scripting etc.) and created a tool named "Pixy". "Pixy" is based on static analysis that detects the malicious activities from the source code of the applications. During the experiments, "Pixy" detected 15 unknown vulnerabilities in three web applications.

Christodorescu and Jha [Chri00] developed in their article a static analysis tool called "SAFE" for detecting malicious executable files. "SAFE" was created to provide a better solution for obfuscation attacks. After the experiments phase, the results showed that "SAFE" detected malicious files better than three familiar Anti-Virus software. All three succeed in detecting the different viruses but when obfuscation code was used, it tricked the Anti-Virus softwares.

In the article [ThSc13] Thesis and Science tried to check if it is possible to detect malicious PDF files which were sent as attachments via email. The author elaborated about malicious file that was sent as attachment, which can also be Microsoft office and EXE program. Through the research, the author used two tools - "PDF Scrutinizer" and "MDScan". Both of the tools use static and dynamic analysis of PDF files. The conclusion of the author was that it is possible to detect malicious PDF files but it is important to keep developing new technologies and tools to detect new malicious attacks.

The article [PrST16] focuses on steganography and how to detect a malware that has been deployed into an image. The authors built a tool for detecting malware from a stegno-image by using steganalysis calculations (chi-square assaults, visual discovery and histogram investigation). The authors concluded that the tool recognize some of the methods of steganography.

In article [HaVi00] Hallaraker and Vigna showed how to detect malicious JavaScript code in the Mozilla browser. They showed two ways to do it - the first is by anomaly detection and the second is by writing a signatures. Anomaly detection is the action of comparing the behavior of the script to a "normal" behavior, while writing signature is comparing the actions of the script to predefined attack scenarios. Example for writing signatures: every script that will open more than 4 windows or if the script set the location of the document with more than one argument, the system can detect and specify the script as malicious. This research shows the importance of using expert features in the detection efforts.

In article [Otsu15] Otsubo presented how experts can help to detect malicious file. By understanding the problem, the way of attack and how the attackers exploit the documents' vulnerabilities, the authors created a tool called "O-checker". In Japan, through 2014, many organizations had received targeted emails with malicious files as attachments. "O-checker" was created to help those organizations detect malicious files like PDF, RTF etc. The base of "O-checker" is that by understanding the structure of the files, it can define exceptions like embedded text after "EOF" or the correctness of the structure of the specific file. Through the experiments, the authors compared "O-checker" performance to Anti-Virus software. "O-checker" detected malicious files with 96.1% of success.

In article [MaGi00] Maiorca and Giacinto showed the simple way, in which attackers can change their malware code and create PDF files that can evade software that used structural features. The attack called "The Mimicry attack" when the hacker mimics a benign PDF structure and change the parts he knows the detection software is not checking. This article empowers the claim that detection efforts must keep developing all the time in order to be unpredictable and hard to bypass.

In article [ANWN07] a comparison between some of those techniques like Random forests, SVM, Neural Networks, Bayesian additive regression trees, logistic regression and Classification and Regression trees. For the experiments step, the author created a data set of malicious and cleaned emails (1171 raw phishing emails and 1718 legitimate emails) and parsed them for testing. After that, they ran all the different models, 30 times with 43 features. The article's results show that Random forests presented the little errors but showed the worst FP ratio. Figure 3 presents the average error for all classifiers and we can see that Random Forest produces the little error. Figure 4 presents FP and FN ratio for all classifiers and we can see that Random Forest produces the worst FP ratio.

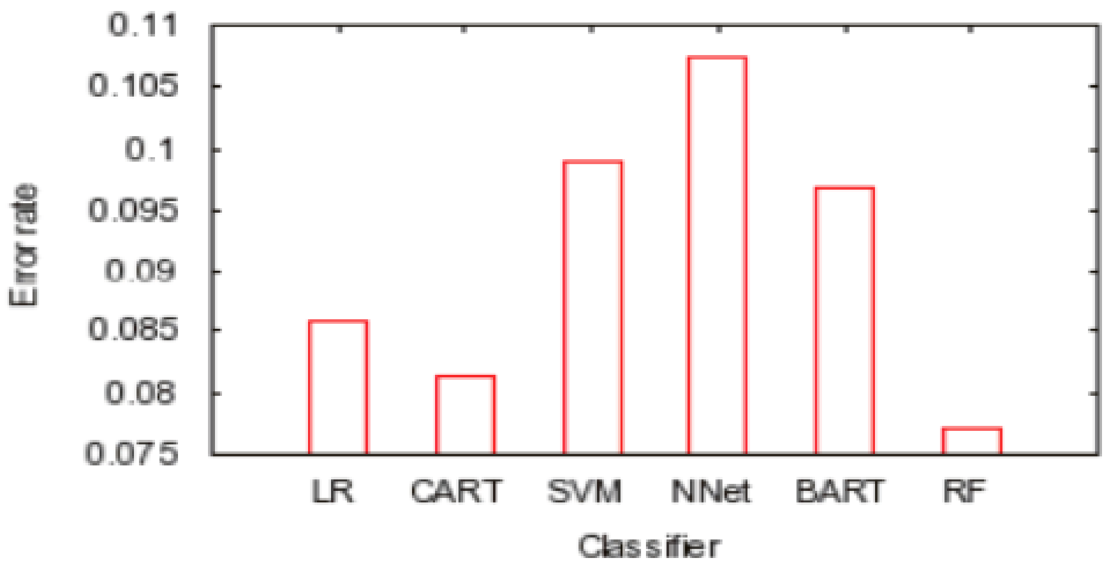
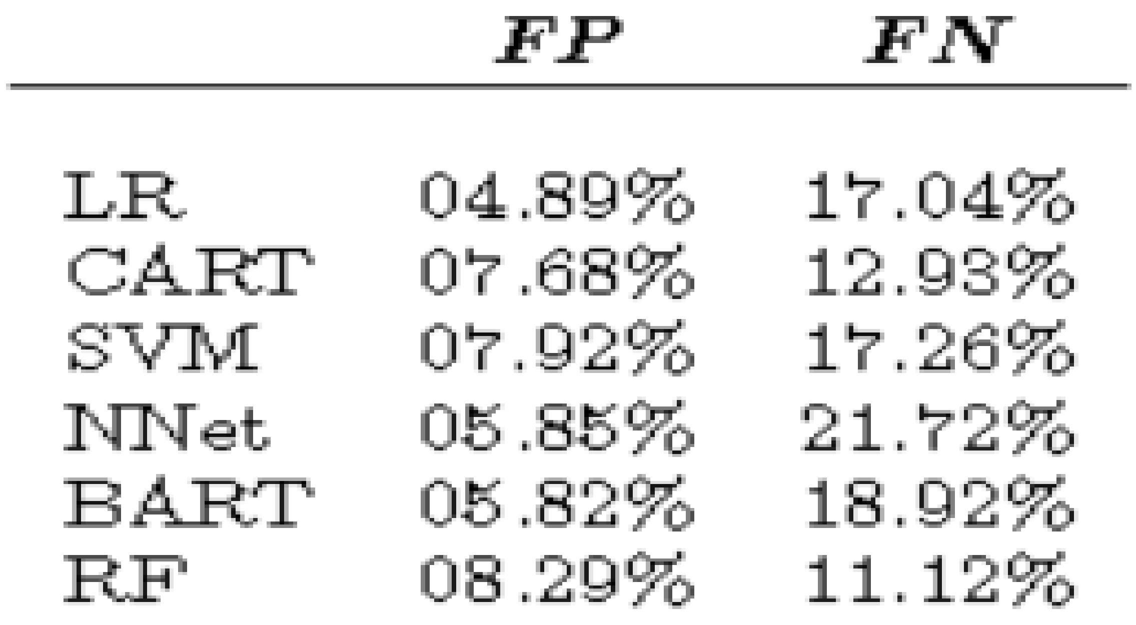


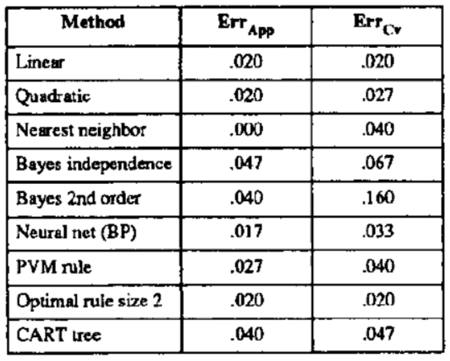
Figure 3. Average $W_{err}(\lambda = 1)$ for all clasifiers



	FP	FN
LR	04.89%	17.04%
CART	07.68%	12.93%
SVM	07.92%	17.26%
NNet	05.85%	21.72%
BART	05.82%	18.92%
RF	08.29%	11.12%

Figure 4. False positive (FP) rate and false negative (FN) rate for all classifiers

In article [WeKa89] Weiss and Kapouleas showed an empirical comparison between Pattern recognition (Linear discriminant, Quadratic discriminant, Nearest neighbor, Bayes independence and Bayes second order), neural nets and machine learning classification methods. To give a verdict, the authors ran four experiments (we will share the results of two of them). The first experiment consists of the Iris Database. Figure 5 presents the errors for all the classifiers for comparison usage while Figure 6 presents the rate of errors only for neural net on Fisher's Iris data.



Method	Err_{App}	Err_{Cv}
Linear	.020	.020
Quadratic	.020	.027
Nearest neighbor	.000	.040
Bayes independence	.047	.067
Bayes 2nd order	.040	.160
Neural net (BP)	.017	.033
PVM rule	.027	.040
Optimal rule size 2	.020	.020
CART tree	.040	.047

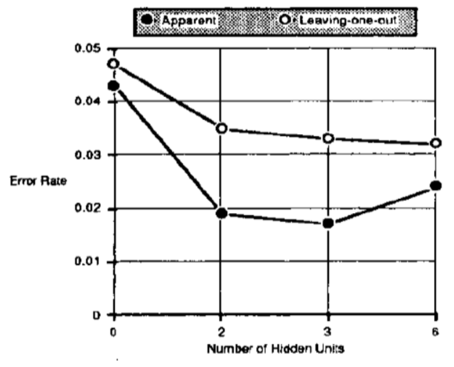
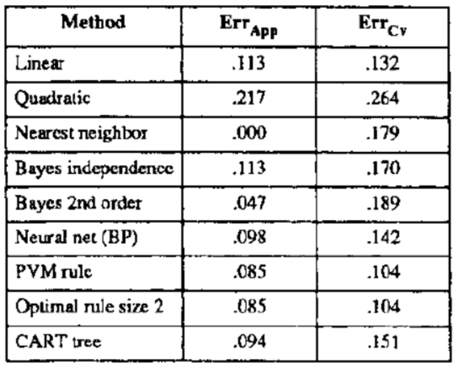


Figure 5.Comparative performance on Fisher's Iris data

Figure 6. Neural net Error rates for Iris data

The second experiment consists of the Appendicitis Patients Database (106 patients and 8 diagnostic tests). Figure 7 presents the errors for all the classifiers for comparison usage while Figure 8 presents the rate of errors only for neural net on Appendicitis data. The authors didn't give a clear verdict, but they showed that before choosing the classifier, it is important to check some of them on the relevant database.



Method	Err_{App}	Err_{Cv}
Linear	.113	.132
Quadratic	.217	.264
Nearest neighbor	.000	.179
Bayes independence	.113	.170
Bayes 2nd order	.047	.189
Neural net (BP)	.098	.142
PVM rule	.085	.104
Optimal rule size 2	.085	.104
CART tree	.094	.151

Figure 7. Comparative performance on Appendicitis data

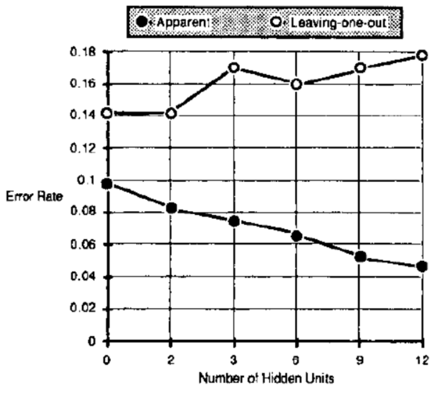
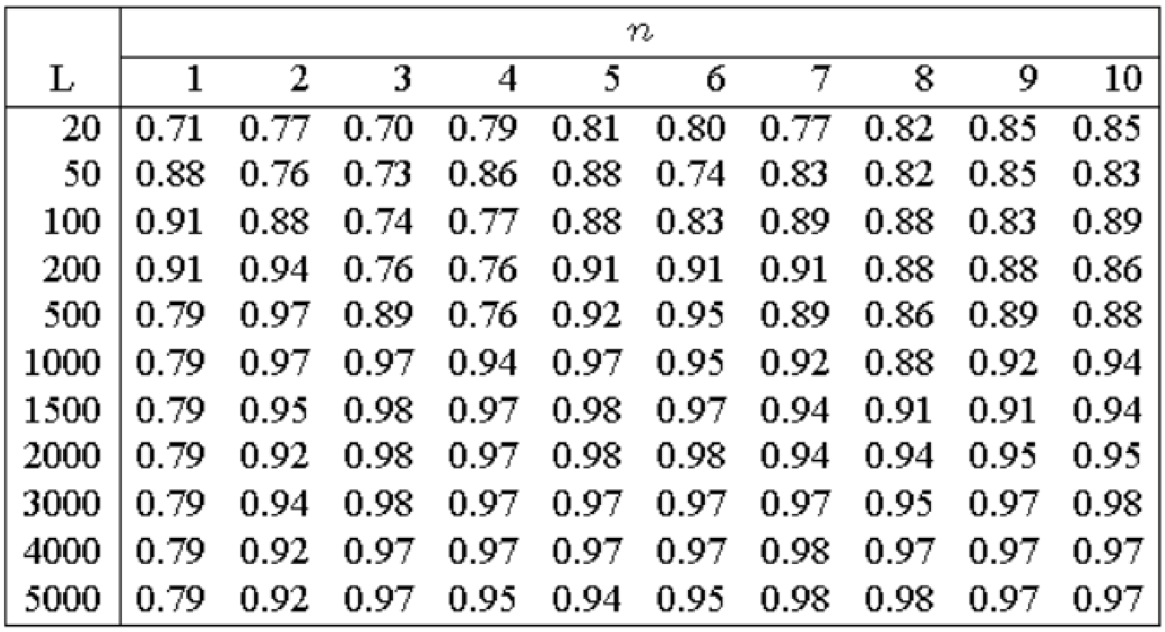


Figure 8. Neural net error rates for Appendicitis data

The article [ACKS00] focused on using the N-gram algorithm for the detection of new malicious code. According to this article, using N-gram has helped to detect strings and substrings that define malicious code and it evaluated the detection of malicious code to 98% at the test set. Figure 9 presents the accuracy for L (profile length) and n (n-gram size). We can see higher accuracy ratio as long as the authors used higher n and L.



L	n									
	1	2	3	4	5	6	7	8	9	10
20	0.71	0.77	0.70	0.79	0.81	0.80	0.77	0.82	0.85	0.85
50	0.88	0.76	0.73	0.86	0.88	0.74	0.83	0.82	0.85	0.83
100	0.91	0.88	0.74	0.77	0.88	0.83	0.89	0.88	0.83	0.89
200	0.91	0.94	0.76	0.76	0.91	0.91	0.91	0.88	0.88	0.86
500	0.79	0.97	0.89	0.76	0.92	0.95	0.89	0.86	0.89	0.88
1000	0.79	0.97	0.97	0.94	0.97	0.95	0.92	0.88	0.92	0.94
1500	0.79	0.95	0.98	0.97	0.98	0.97	0.94	0.91	0.91	0.94
2000	0.79	0.92	0.98	0.97	0.98	0.98	0.94	0.94	0.95	0.95
3000	0.79	0.94	0.98	0.97	0.97	0.97	0.97	0.95	0.97	0.98
4000	0.79	0.92	0.97	0.97	0.97	0.97	0.98	0.97	0.97	0.97
5000	0.79	0.92	0.97	0.95	0.94	0.95	0.98	0.98	0.97	0.97

Figure 9. 3-fold cross-validation accuracy

Another article [SPDB00] dealed with N-gram technique for detection. The authors collected 149,882 malicious files and 4934 benign files and divided them into train and test set. The authors tried to find the optimal size of n-gram, the number of nearest neighbors for the knn algorithm and the limit that marks the nature of the unknown instance as malware. We can see the results (detection ratio and False Positive ratio) in the diagram Figure 10. Using the parameters: n-gram size is 6, the number of nearest neighbors for the knn algorithm is 7 and the limit that marks the nature of the unknown instance as malware is 2, provided the optimal combination between the maximum detection ratio and lowest False Positive ratio.

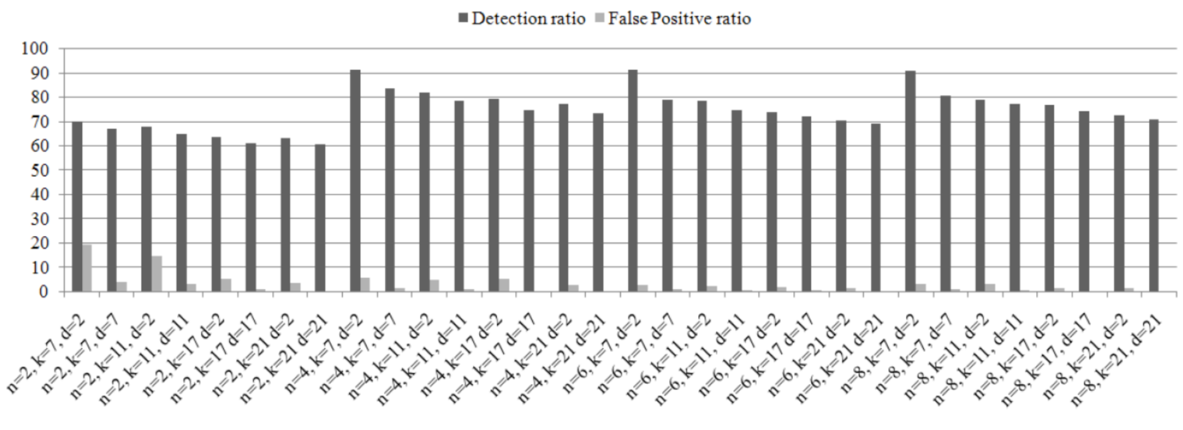


Figure 10. Detection and false positive rate

In the article [CNRE16] Cohen, Nissim, Rokach and Elovici used a structural feature extraction methodology for the detection malicious office files. Every file was unzipped and every sub file was entered as a path in the file. Thus, every file was a unique list with different paths. By taking a big database (16,108 benign and 830 malicious .docx files), they ran four experiments. The first one includes using machine learning with different classifiers (J48, Random forest, Logistic regression etc.) in a 10-fold cross-validation. The second experiment includes checking the paths without numbers. The third experiment includes compression using N-gram and the fourth includes comparison of the detection rate with Anti-Virus softwares. We can see some the results of couple of those experiments. Figure 11 presents the comparison between the different AUC of the classifiers (from the top of 200 on, Random forest of 500 trees shows the better performances). Figure 12 presents the True-Positive Ratio of SFEM methods relatively to other Anti-Virus softwares (SFEM presented the best ratio of 97% TP):

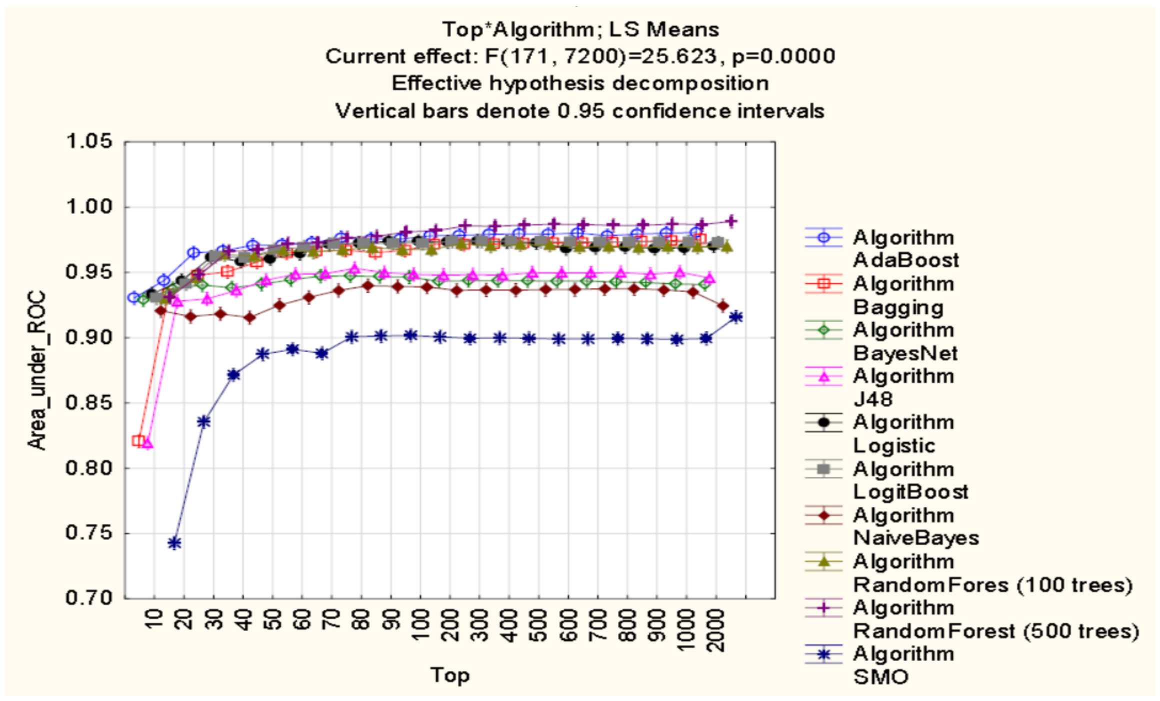


Figure 11.Comparasion of the average AUC of the selected classifiers on different top-features datasets

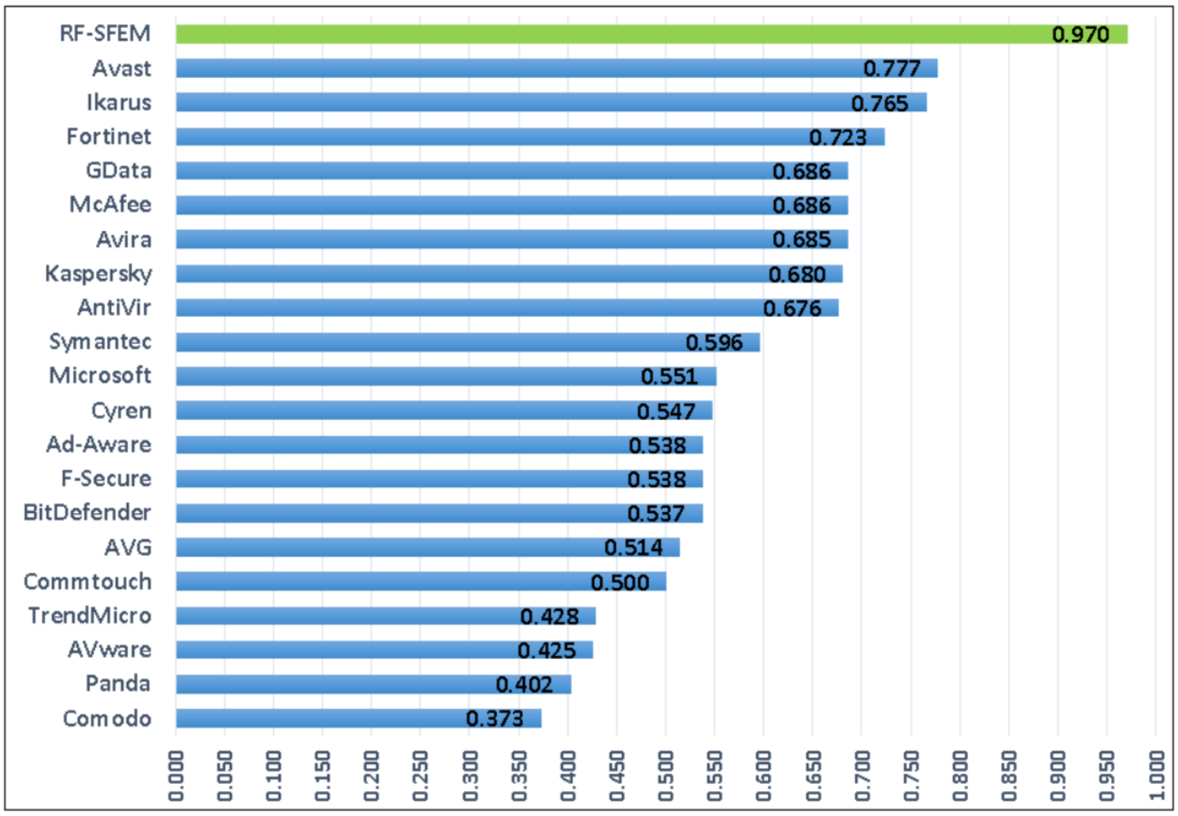


Figure 12.The TPR of RF-SFEM method compared to anti-viruses commonly used by organizations

The results showed that using machine learning improve the detection rate in very high percentages. By using the random forest classifier, the authors achieved an AUC of 99.12%, True positive rate of 97% and false positive rate of 4.9%.

Nissim, Cohen, Glezer and Elovici [NCGE14] displayed how to implement the same machine learning methods from the office file to PDF. The research used Feature extraction on the PDF structure. Creation of paths from top to bottom and every path can be a feature.

Microsoft RTF specification 1.9.1 [Serv08], for thorough understanding of the file structure. The file consists every details and examples of every Group, Control word etc. Also, the document explains the different between the header and the documents group.

The book "JPEG still image data compression standard" [PeMi93] for thorough understanding of the file structure. In the book we found out every detail about the way JPEG compression is working, the vary markers.

V. METHODS

In our project, we are detecting malicious files with machine learning classifiers trained on datasets containing features generated by the feature extraction methods that we will present. Feature extraction is a method that takes data and divides it to parts (so called features). The division depends on the data and the way it's built. (In Sections "Background" and "JPEG File Structure" we elaborated about the patterns in JPEG file).

In this section, we will present two different methods for feature extraction. The first method is a predefined, knowledge-based, finite set of features, based on known characteristics of malicious and benign files. The second method is feature extraction methodology based on the structure of JPEG file presented earlier. In addition we will elaborate on the feature selection methods and feature representation method. We will also present the data collection we used for evaluation and describe the process of dataset creation.

To examine our methods, we collected both malicious and benign JPEG files from *VirusTotal* in addition to benign files from Google. Our collection is composed of 819 malicious and 6,279 benign files. To sum it up, the data collection consists 7098 JPEG files. The percentages are approximately 0.11 malicious and approximately 0.89 benign files.

A. Data Creation – Structural

Visually, you can find the whole process of the feature extraction in Figure 13.

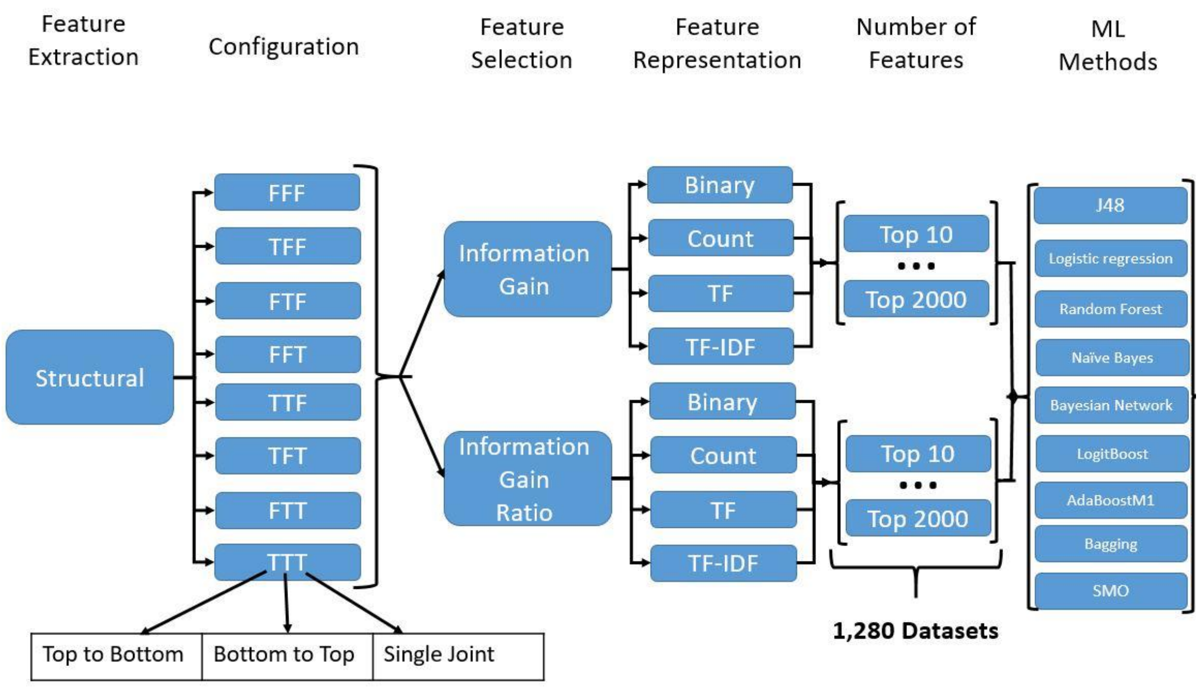
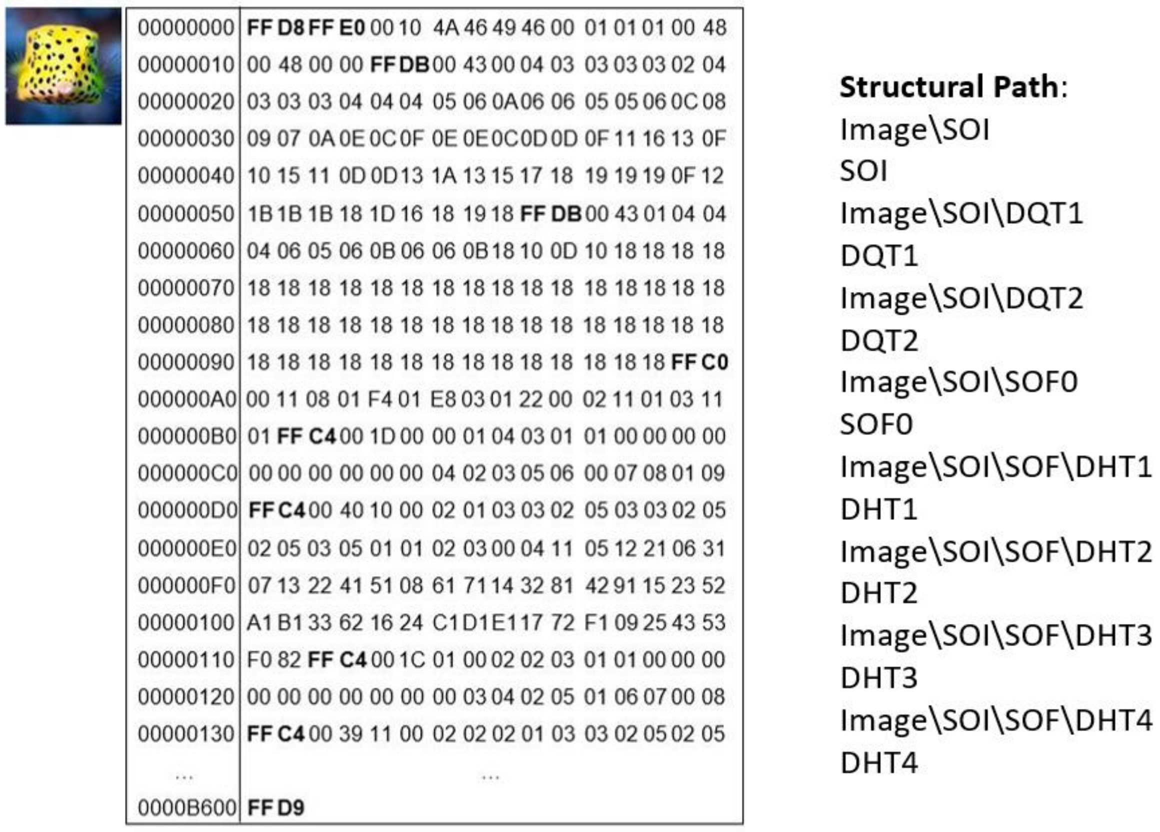


Figure 13. The Whole Structural Process of Feature Extraction

1. *Feature Extraction*

The purpose of feature extraction process is to extract discriminative features from the examined element. The extracted features will be leveraged using machine learning algorithms for the detection of the correct class of the element.

As we mentioned in the start of the section, the partition to features depends on the data and the way it's built. We found out that JPEG files have a hierarchy structure thus, we decided to use this fact to create a strings of features. Every marker starts a feature string (top down configuration) and also, it is a feature by itself (single joint configuration). Every following marker is adding to the feature string and also became a feature by itself. In this way, when the parser finishes extracting the features from a JPEG file, we get every marker as a feature and features that are constructed by series of control words. For example: Image/SOI, SOI, Image/SOI/DQT1, DQT1 are all features. Moreover, we decided to create another set of features by traversing the hierarchical structure of the file from bottom to top (bottom to top configuration). Thus, we collected all the information and unique parts of JPEG file we could get to indicate if the file is malicious. As part of the research, for each configuration of the eight extraction configuration (**Error! Reference source not found.**) we created a dataset of unique features collected from all the JPEG files and their number of occurrences in each file in our collection.



```
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 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
```

Structural Path:
Image\SOI
SOI
Image\SOI\DQT1
DQT1
Image\SOI\DQT2
DQT2
Image\SOI\SOF0
SOF0
Image\SOI\SOF\DHT1
DHT1
Image\SOI\SOF\DHT2
DHT2
Image\SOI\SOF\DHT3
DHT3
Image\SOI\SOF\DHT4
DHT4

Figure 14: Structural Path of JPEG

2. *Feature Selection*

There are three main approaches for feature selection: *filter methods*, *wrapper methods,* and *embedded methods*. *Filter methods* use a measure to evaluate the correlation of each individual feature to the class (malicious/benign). After applying a filter on a dataset, each feature gets a rank which quantifies its expected contribution in the classification task, from which the features with the top X ranks are used to train machine learning algorithms. *Wrapper methods* conduct a search for a good subset of features using the learning algorithm itself as part of the evaluation function. *Embedded methods* perform feature selection as part of the learning procedure and are usually specific to given learning algorithms (i.e., classification trees, etc.).

Since the total number of unique features extracted from our collection is very large, a filter method best fits our needs for a number of reasons: it allows us to select the top X prominent features; it scales easily to high dimensional datasets; and it is computationally simple, fast, and independent of the classification algorithm, thus, enabling us to compare the performances of the different classification algorithms on the same subset of features. We rejected *wrapper methods* for two reasons. This first is because they depend on a classifier in order to select a good subset of features, and each classifier would select a different subset. The second reason is that the process of searching a subset of features in a very high dimensional dataset such as ours is too heavy. We rejected *embedded methods,* since this approach depends on the classifier.

Although we choose the *filter method*, we are aware of the disadvantages. The first disadvantage is that with this approach, the potential contribution from the interaction between the features is neglected. For example, when brought together, a subset of features might contribute significantly to the classification task. However, individual features within the subset might obtain a low rank, and thus may not be included in the top X features. The second disadvantage is that when using the top X features ranked by a specific filter, one might obtain sets of features that contribute equally to the discrimination between classes. However, it is more efficient to use only a single feature from each set and thereby accommodate other features with different contributions.

In this phase we used two methods. The first is information gain (IG) which evaluates the worth of an attribute by measuring how the attributes contributes to discriminate between malicious and benign class. The second is information gain ratio (GR). GR is an extension of IG, which normalizes the value of IG using the so called split info. We used the two methods of selection that give a rank to each feature and sort them according to their prominence to choose the top X ranked features.

3. *Feature Representation*

Feature representation determines how the presence (and importance) of a feature in a file is represented. We used four feature representation methods: Binary, Count, TF and TF-IDF. Binary representation is used to indicate the presence (1) or absence (0) of a feature within the file. TF shows the term frequency in the file normalized by dividing its frequency by the most frequent term in the document. TFIDF is a weighting method often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a term is to a document in a collection. It combines the frequency of a term in the document (TF) and its frequency in the document collection, denoted by DF. The definition of TFIDF is multiplying the term frequency (TF) by the $\mathrm{IDF} = \log_2(\mathrm{N}/\mathrm{DF})$, where N is the number of documents in the entire collection and DF is the number of files in which the term appears. In our case a term is a feature extracted from a JPEG file.

4. *Number of Top Selected Features*

In order to determine how the number of top selected features affects the detection rates, we also took into consideration the top subsets of the most prominent features. We created twenty datasets using the 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1,000, and 2,000 most prominent features selected by a feature selection method.

At the end of the process, we hold 20 datasets (Top 10, 20 … 1000, 2000) for each representation method (binary, count, TF and TFIDF), for each two feature selection method (information gain and information gain ratio) and for each of eight feature extraction configuration method (FFF, TFF, FTF, FFT, TTF, TFT, FTT and TTT). To sum up, we hold 1,280 datasets. We applied several machine learning classifiers on each of these datasets.

B. Data Creation – Knowledge-Based

1. Feature Extraction

The purpose of feature extraction process is to extract discriminative features from the examined element. The extracted features will be leveraged using machine learning algorithms for the detection of the correct class of the element.

The creation of knowledge-based features demands thorough understanding of former attacks in which JPEG files were used, exploited vulnerabilities in JPEG files and the structure of JPEG file (the role of each marker, hierarchy of the file etc.). Table 11 presents the list of features we implemented for the JPEG, including the feature's name, the element of the file that the feature is associated with, the description, feature type (simple, attack-based or expert), data type (binary, integer, etc.).

ID	Feature Name	Element	Description	Feature Type	Data Type
1	File_size	File	Checking the JPEG size and detect suspicious values (0 kb)	Simple	Integer
2	File_extension	File	Checking if it's a JPEG file (starting xFFD8)	Attack-based	Integer
3	File_marker_num	File	Total amount of markers in the image	Simple	Integer
4	Marker_APP0_num	Marker	Total amount of APP0 (0xFFE0) in the image	Simple	Integer
5	Marker_APP1_num	Marker	Total amount of APP1 (0xFFE1) in the image	Simple	Integer
6	Marker_APP2_num	Marker	Total amount of APP2 (0xFFE2) in the image	Simple	Integer
7	Marker_APP3_num	Marker	Total amount of APP3 (0xFFE3) in the image	Simple	Integer
8	Marker_APP4_num	Marker	Total amount of APP4 (0xFFE4) in the image	Simple	Integer
9	Marker_APP5_num	Marker	Total amount of APP5 (0xFFE5) in the image	Simple	Integer
10	Marker_APP6_num	Marker	Total amount of APP6 (0xFFE6) in the image	Simple	Integer
11	Marker_APP7_num	Marker	Total amount of APP7 (0xFFE7) in the image	Simple	Integer

12	Marker_APP8_num	Marker	Total amount of APP8 (0xFFE8) in the image	Simple	Integer
13	Marker_APP9_num	Marker	Total amount of APP9 (0xFFE9) in the image	Simple	Integer
14	Marker_APP10_num	Marker	Total amount of APP10 (0xFFEA) in the image	Simple	Integer
15	Marker_APP11_num	Marker	Total amount of APP11 (0xFFEB) in the image	Simple	Integer
16	Marker_APP12_num	Marker	Total amount of APP12 (0xFFEC) in the image	Simple	Integer
17	Marker_APP13_num	Marker	Total amount of APP13 (0xFFED) in the image	Simple	Integer
18	Marker_APP14_num	Marker	Total amount of APP14 (0xFFEE) in the image	Simple	Integer
19	Marker_APP15_num	Marker	Total amount of APP15 (0xFFEF) in the image	Simple	Integer
20	Marker_COM_num	Marker	Total amount of COM (0xFFFE) in the image	Simple	Integer
21	Marker_DAC_num	Marker	Total amount of DAC (0xFFCC) in the image	Simple	Integer
22	Marker_DHP_num	Marker	Total amount of DHP (0xFFDE) in the image	Simple	Integer
23	Marker_DHT_num	Marker	Total amount of DHT (0xFFC4) in the image	Simple	Integer
24	Marker_DNL_num	Marker	Total amount of DNL (0xFFDC) in the image	Simple	Integer
25	Marker_DQT_num	Marker	Total amount of DQT (0xFFDB) in the image	Simple	Integer
26	Marker_DRI_num	Marker	Total amount of DRI (0xFFDD) in the image	Simple	Integer
27	Marker_EOI_num	Marker	Total amount of EOI (0xFFD9) in the image	Simple	Integer
28	Marker_EXP_num	Marker	Total amount of EXP (0xFFDF) in the image	Simple	Integer
29	Marker_JPG_num	Marker	Total amount of JPG (0xFFC8) in the image	Simple	Integer
30	Marker_JPG0_num	Marker	Total amount of JPG0 (0xFFF0) in the image	Simple	Integer
31	Marker_JPG1_num	Marker	Total amount of JPG1 (0xFFF1) in the image	Simple	Integer

32	Marker_JPG2_num	Marker	Total amount of JPG2 (0xFFF2) in the image	Simple	Integer
33	Marker_JPG3_num	Marker	Total amount of JPG3 (0xFFF3) in the image	Simple	Integer
34	Marker_JPG4_num	Marker	Total amount of JPG4 (0xFFF4) in the image	Simple	Integer
35	Marker_JPG5_num	Marker	Total amount of JPG5 (0xFFF5) in the image	Simple	Integer
36	Marker_JPG6_num	Marker	Total amount of JPG6 (0xFFF6) in the image	Simple	Integer
37	Marker_JPG7_num	Marker	Total amount of JPG7 (0xFFF7) in the image	Simple	Integer
38	Marker_JPG8_num	Marker	Total amount of JPG8 (0xFFF8) in the image	Simple	Integer
39	Marker_JPG9_num	Marker	Total amount of JPG9 (0xFFF9) in the image	Simple	Integer
40	Marker_JPG10_num	Marker	Total amount of JPG10 (0xFFFA) in the image	Simple	Integer
41	Marker_JPG11_num	Marker	Total amount of JPG11 (0xFFFB) in the image	Simple	Integer
42	Marker_JPG12_num	Marker	Total amount of JPG12 (0xFFFC) in the image	Simple	Integer
43	Marker_JPG13_num	Marker	Total amount of JPG13 (0xFFFD) in the image	Simple	Integer
44	Marker_RST0_num	Marker	Total amount of RST0 (0xFFD0) in the image	Simple	Integer
45	Marker_RST1_num	Marker	Total amount of RST1 (0xFFD1) in the image	Simple	Integer
46	Marker_RST2_num	Marker	Total amount of RST2 (0xFFD2) in the image	Simple	Integer
47	Marker_RST3_num	Marker	Total amount of RST3 (0xFFD3) in the image	Simple	Integer
48	Marker_RST4_num	Marker	Total amount of RST4 (0xFFD4) in the image	Simple	Integer
49	Marker_RST5_num	Marker	Total amount of RST5 (0xFFD5) in the image	Simple	Integer
50	Marker_RST6_num	Marker	Total amount of RST6 (0xFFD6) in the image	Simple	Integer
51	Marker_RST7_num	Marker	Total amount of RST7 (0xFFD7) in the image	Simple	Integer

52	Marker_SOF0_num	Marker	Total amount of SOF0 (0xFFC0) in the image	Simple	Integer
53	Marker_SOF1_num	Marker	Total amount of SOF1 (0xFFC1) in the image	Simple	Integer
54	Marker_SOF2_num	Marker	Total amount of SOF2 (0xFFC2) in the image	Simple	Integer
55	Marker_SOF3_num	Marker	Total amount of SOF3 (0xFFC3) in the image	Simple	Integer
56	Marker_SOF5_num	Marker	Total amount of SOF5 (0xFFC5) in the image	Simple	Integer
57	Marker_SOF6_num	Marker	Total amount of SOF6 (0xFFC5) in the image	Simple	Integer
58	Marker_SOF7_num	Marker	Total amount of SOF7 (0xFFC5) in the image	Simple	Integer
59	Marker_SOF9_num	Marker	Total amount of SOF9 (0xFFC9) in the image	Simple	Integer
60	Marker_SOF10_num	Marker	Total amount of SOF10 (0xFFCA) in the image	Simple	Integer
61	Marker_SOF11_num	Marker	Total amount of SOF11 (0xFFCB) in the image	Simple	Integer
62	Marker_SOF13_num	Marker	Total amount of SOF13 (0xFFCD) in the image	Simple	Integer
63	Marker_SOF14_num	Marker	Total amount of SOF14 (0xFFCE) in the image	Simple	Integer
64	Marker_SOF15_num	Marker	Total amount of SOF15 (0xFFCF) in the image	Simple	Integer
65	Marker_SOI_num	Marker	Total amount of SOI (0xFFD8) in the image	Simple	Integer
66	Marker_SOS_num	Marker	Total amount of SOS (0xFFDA) in the image	Simple	Integer
67	Marker_TEM_num	Marker	Total amount of TEM (0xFF01) in the image	Simple	Integer
68	Marker_SOI_EOI_dif	Marker	Difference between amount of SOI and amount of EOI.	Expert	Integer
69	Marker_EOI_illegal_content_atfer_num	Marker	Number of illegal content after EOI. Whether between EOI and SOI or after SOI. {SOI EOI} illegal content {SOI EOI}. {SOI EOI} illegal content.	Expert	Integer
70	Marker_APP0_size_max	Marker	Size of APP0 (0xFFE0) in the image [2 bytes after the marker]. If there are more than	Expert	Integer

			one – represents the max size.		
71	Marker_APP0_size_incorrect_num	Marker	Number of APP0 (0xFFE0) in the image there size is illegal – 0 or 1	Expert	Integer
72	Marker_APP0_size_declared_incorrect_num	Marker	Number of APP0 markers declare incorrect size found in the image.	Expert	Integer
73	Marker_APP1_size_max	Marker	Size of APP1 (0xFFE1) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
74	Marker_APP1_size_incorrect_num	Marker	Number of APP1 (0xFFE1) in the image there size is illegal – 0 or 1	Expert	Integer
75	Marker_APP1_size_declared_incorrect_num	Marker	Number of APP1 markers declare incorrect size found in the image.	Expert	Integer
76	Marker_APP2_size_max	Marker	Size of APP2 (0xFFE2) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
77	Marker_APP2_size_incorrect_num	Marker	Number of APP2 (0xFFE2) in the image there size is illegal – 0 or 1	Expert	Integer
78	Marker_APP2_size_declared_incorrect_num	Marker	Number of APP2 markers declare incorrect size found in the image.	Expert	Integer
79	Marker_APP3_size_max	Marker	Size of APP3 (0xFFE3) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
80	Marker_APP3_size_incorrect_num	Marker	Number of APP3 (0xFFE3) in the image there size is illegal – 0 or 1	Expert	Integer
81	Marker_APP3_size_declared_incorrect_num	Marker	Number of APP3 markers declare incorrect size found in the image.	Expert	Integer
82	Marker_APP4_size_max	Marker	Size of APP4 (0xFFE4) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
83	Marker_APP4_size_incorrect_num	Marker	Number of APP4 (0xFFE4) in the image there size is illegal – 0 or 1	Expert	Integer
84	Marker_APP4_size_declared_incorrect_num	Marker	Number of APP4 markers declare incorrect size found in the image.	Expert	Integer
85	Marker_APP5_size_max	Marker	Size of APP5 (0xFFE5) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer

86	Marker_APP5_size_incorrect_num	Marker	Number of APP5 (0xFFE5) in the image there size is illegal – 0 or 1	Expert	Integer
87	Marker_APP5_size_declared_incorrect_num	Marker	Number of APP5 markers declare incorrect size found in the image.	Expert	Integer
88	Marker_APP6_size_max	Marker	Size of APP6 (0xFFE6) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
89	Marker_APP6_size_incorrect_num	Marker	Number of APP6 (0xFFE6) in the image there size is illegal – 0 or 1	Expert	Integer
90	Marker_APP6_size_declared_incorrect_num	Marker	Number of APP6 markers declare incorrect size found in the image.	Expert	Integer
91	Marker_APP7_size_max	Marker	Size of APP7 (0xFFE7) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
92	Marker_APP7_size_incorrect_num	Marker	Number of APP7 (0xFFE7) in the image there size is illegal – 0 or 1	Expert	Integer
93	Marker_APP7_size_declared_incorrect_num	Marker	Number of APP7 markers declare incorrect size found in the image.	Expert	Integer
94	Marker_APP8_size_max	Marker	Size of APP8 (0xFFE8) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
95	Marker_APP8_size_incorrect_num	Marker	Number of APP8 (0xFFE8) in the image there size is illegal – 0 or 1	Expert	Integer
96	Marker_APP8_size_declared_incorrect_num	Marker	Number of APP8 markers declare incorrect size found in the image.	Expert	Integer
97	Marker_APP9_size_max	Marker	Size of APP9 (0xFFE9) in the image [2 bytes after the marker]. If there are more than one –represents the max size.	Expert	Integer
98	Marker_APP9_size_incorrect_num	Marker	Number of APP9 (0xFFE9) in the image there size is illegal – 0 or 1	Expert	Integer
99	Marker_APP9_size_declared_incorrect_num	Marker	Number of APP9 markers declare incorrect size found in the image.	Expert	Integer
100	Marker_APP10_size_max	Marker	Size of APP10 (0xFFEA) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
101	Marker_APP10_size_incorrect_num	Marker	Number of APP10 (0xFFEA) in the image there size is illegal – 0 or 1	Expert	Integer

102	Marker_APP10_size_declared_incorrect_num	Marker	Number of APP10 markers declare incorrect size found in the image.	Expert	Integer
103	Marker_APP11_size_max	Marker	Size of APP11 (0xFFEB) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
104	Marker_APP11_size_incorrect_num	Marker	Number of APP11 (0xFFEB) in the image there size is illegal – 0 or 1	Expert	Integer
105	Marker_APP11_size_declared_incorrect_num	Marker	Number of APP11 markers declare incorrect size found in the image.	Expert	Integer
106	Marker_APP12_size_max	Marker	Size of APP12 (0xFFEC) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
107	Marker_APP12_size_incorrect_num	Marker	Number of APP12 (0xFFEC) in the image there size is illegal – 0 or 1	Expert	Integer
108	Marker_APP12_size_declared_incorrect_num	Marker	Number of APP12 markers declare incorrect size found in the image.	Expert	Integer
109	Marker_APP13_size_max	Marker	Size of APP13 (0xFFED) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
110	Marker_APP13_size_incorrect_num	Marker	Number of APP13 (0xFFED) in the image there size is illegal – 0 or 1	Expert	Integer
111	Marker_APP13_size_declared_incorrect_num	Marker	Number of APP13 markers declare incorrect size found in the image.	Expert	Integer
112	Marker_APP14_size_max	Marker	Size of APP14 (0xFFEE) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
113	Marker_APP14_size_incorrect_num	Marker	Number of APP14 (0xFFEE) in the image there size is illegal – 0 or 1	Expert	Integer
114	Marker_APP14_size_declared_incorrect_num	Marker	Number of APP14 markers declare incorrect size found in the image.	Expert	Integer
115	Marker_APP15_size_max	Marker	Size of APP15 (0xFFEF) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
116	Marker_APP15_size_incorrect_num	Marker	Number of APP15 (0xFFEF) in the image there size is illegal – 0 or 1	Expert	Integer
117	Marker_APP15_size_declared_incorrect_num	Marker	Number of APP15 markers declare incorrect size found in the image.	Expert	Integer

118	Marker_COM_size_max	Marker	Size of COM (0xFFFE) in the image [2 bytes after the marker] . If there are more than one – represents the max size.	Expert	Integer
119	Marker_COM_size_incorrect_num	Marker	Number of COM (0xFFFE) in the image there size is illegal – 0 or 1	Attack-based	Integer
120	Marker_COM_size_declared_incorrect_num	Marker	Number of COM markers declare incorrect size found in the image.	Expert	Integer
121	Marker_DAC_size_max	Marker	Size of DAC (0xFFCC) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
122	Marker_DAC_size_incorrect_num	Marker	Number of DAC (0xFFCC) in the image there size is illegal – 0 or 1	Expert	Integer
123	Marker_DAC_size_declared_incorrect_num	Marker	Number of DAC markers declare incorrect size found in the image.	Expert	Integer
124	Marker_DHP_size_max	Marker	Size of DHP (0xFFDE) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
125	Marker_DHP_size_incorrect_num	Marker	Number of DHP (0xFFDE) in the image there size is illegal – 0 or 1	Expert	Integer
126	Marker_DHP_size_declared_incorrect_num	Marker	Number of DHP markers declare incorrect size found in the image.	Expert	Integer
127	Marker_DHT_size_max	Marker	Size of DHT (0xFFC4) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
128	Marker_DHT_size_incorrect_num	Marker	Number of DHT (0xFFC4) in the image there size is illegal – 0 or 1	Expert	Integer
129	Marker_DHT_size_declared_incorrect_num	Marker	Number of DHT markers declare incorrect size found in the image.	Expert	Integer
130	Marker_DNL_size_max	Marker	Size of DNL (0xFFDC) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
131	Marker_DNL_size_incorrect_num	Marker	Number of DNL (0xFFDC) in the image there size is illegal – 0 or 1	Expert	Integer
132	Marker_DNL_size_declared_incorrect_num	Marker	Number of DNL markers declare incorrect size found in the image.	Expert	Integer
133	Marker_DQT_size_max	Marker	Size of DQT (0xFFDB) in the image [2 bytes after the marker]. If there are more than	Expert	Integer

			one – represents the max size.		
134	Marker_DQT_size_incorrect_num	Marker	Number of DQT (0xFFDB) in the image there size is illegal – 0 or 1	Expert	Integer
135	Marker_DQT_size_declared_incorrect_num	Marker	Number of DQT markers declare incorrect size found in the image.	Expert	Integer
136	Marker_DRI_size_max	Marker	Size of DRI (0xFFDD) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
137	Marker_DRI_size_incorrect_num	Marker	Number of DRI (0xFFDD) in the image there size is illegal – 0 or 1	Expert	Integer
138	Marker_DRI_size_declared_incorrect_num	Marker	Number of DRI markers declare incorrect size found in the image.	Expert	Integer
139	Marker_EXP_size_max	Marker	Size of EXP (0xFFDF) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
140	Marker_EXP_size_incorrect_num	Marker	Number of EXP (0xFFDF) in the image there size is illegal – 0 or 1	Expert	Integer
141	Marker_EXP_size_declared_incorrect_num	Marker	Number of EXP markers declare incorrect size found in the image.	Expert	Integer
142	Marker_JPG_size_max	Marker	Size of JPG (0xFFC8) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
143	Marker_JPG_size_incorrect_num	Marker	Number of JPG (0xFFC8) in the image there size is illegal – 0 or 1	Expert	Integer
144	Marker_JPG_size_declared_incorrect_num	Marker	Number of JPG markers declare incorrect size found in the image.	Expert	Integer
145	Marker_JPG0_size_max	Marker	Size of JPG0 (0xFFF0) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
146	Marker_JPG0_size_incorrect_num	Marker	Number of JPG0 (0xFFF0) in the image there size is illegal – 0 or 1	Expert	Integer
147	Marker_JPG0_size_declared_incorrect_num	Marker	Number of JPG0 markers declare incorrect size found in the image.	Expert	Integer
148	Marker_JPG1_size_max	Marker	Size of JPG1 (0xFFF1) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer

149	Marker_ JPG1_size_incorrect_num	Marker	Number of JPG1 (0xFFF1) in the image there size is illegal – 0 or 1	Expert	Integer
150	Marker_ JPG1_size_declared_incorrect_num	Marker	Number of JPG1 markers declare incorrect size found in the image.	Expert	Integer
151	Marker_JPG2_size_max	Marker	Size of JPG2 (0xFFF2) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
152	Marker_ JPG2_size_incorrect_num	Marker	Number of JPG2 (0xFFF2) in the image there size is illegal – 0 or 1	Expert	Integer
153	Marker_ JPG2_size_declared_incorrect_num	Marker	Number of JPG2 markers declare incorrect size found in the image.	Expert	Integer
154	Marker_JPG3_size_max	Marker	Size of JPG3 (0xFFF3) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
155	Marker_ JPG3_size_incorrect_num	Marker	Number of JPG3 (0xFFF3) in the image there size is illegal – 0 or 1	Expert	Integer
156	Marker_ JPG3_size_declared_incorrect_num	Marker	Number of JPG3 markers declare incorrect size found in the image.	Expert	Integer
157	Marker_JPG4_size_max	Marker	Size of JPG4 (0xFFF4) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
158	Marker_ JPG4_size_incorrect_num	Marker	Number of JPG4 (0xFFF4) in the image there size is illegal – 0 or 1	Expert	Integer
159	Marker_ JPG4_size_declared_incorrect_num	Marker	Number of JPG4 markers declare incorrect size found in the image.	Expert	Integer
160	Marker_JPG5_size_max	Marker	Size of JPG5 (0xFFF5) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
161	Marker_ JPG5_size_incorrect_num	Marker	Number of JPG5 (0xFFF5) in the image there size is illegal – 0 or 1	Expert	Integer
162	Marker_ JPG5_size_declared_incorrect_num	Marker	Number of JPG5 markers declare incorrect size found in the image.	Expert	Integer
163	Marker_JPG6_size_max	Marker	Size of JPG6 (0xFFF6) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
164	Marker_ JPG6_size_incorrect_num	Marker	Number of JPG6 (0xFFF6) in the image there size is illegal – 0 or 1	Expert	Integer

165	Marker_ JPG6_size_declared_incorrect_num	Marker	Number of JPG6 markers declare incorrect size found in the image.	Expert	Integer
166	Marker_JPG7_size_max	Marker	Size of JPG7 (0xFFF7) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
167	Marker_ JPG7_size_incorrect_num	Marker	Number of JPG7 (0xFFF7) in the image there size is illegal – 0 or 1	Expert	Integer
168	Marker_ JPG7_size_declared_incorrect_num	Marker	Number of JPG7 markers declare incorrect size found in the image.	Expert	Integer
169	Marker_JPG8_size_max	Marker	Size of JPG8 (0xFFF8) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
170	Marker_ JPG8_size_incorrect_num	Marker	Number of JPG8 (0xFFF8) in the image there size is illegal – 0 or 1	Expert	Integer
171	Marker_ JPG8_size_declared_incorrect_num	Marker	Number of JPG8 markers declare incorrect size found in the image.	Expert	Integer
172	Marker_JPG9_size_max	Marker	Size of JPG9 (0xFFF9) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
173	Marker_ JPG9_size_incorrect_num	Marker	Number of JPG9 (0xFFF9) in the image there size is illegal – 0 or 1	Expert	Integer
174	Marker_ JPG9_size_declared_incorrect_num	Marker	Number of JPG9 markers declare incorrect size found in the image.	Expert	Integer
175	Marker_JPG10_size_max	Marker	Size of JPG10 (0xFFFA) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
176	Marker_ JPG10_size_incorrect_num	Marker	Number of JPG10 (0xFFFA) in the image there size is illegal – 0 or 1	Expert	Integer
177	Marker_ JPG10_size_declared_incorrect_num	Marker	Number of JPG10 markers declare incorrect size found in the image.	Expert	Integer
178	Marker_JPG11_size_max	Marker	Size of JPG11 (0xFFFB) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
179	Marker_ JPG11_size_incorrect_num	Marker	Number of JPG11 (0xFFFB) in the image there size is illegal – 0 or 1	Expert	Integer
180	Marker_ JPG11_size_declared_incorrect_num	Marker	Number of JPG11 markers declare incorrect size found in the image.	Expert	Integer

181	Marker_JPG12_size_max	Marker	Size of JPG12 (0xFFFC) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
182	Marker_ JPG12_size_incorrect_num	Marker	Number of JPG12 (0xFFFC) in the image there size is illegal – 0 or 1	Expert	Integer
183	Marker_ JPG12_size_declared_incorrect_num	Marker	Number of JPG12 markers declare incorrect size found in the image.	Expert	Integer
184	Marker_JPG13_size_max	Marker	Size of JPG13 (0xFFFD) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
185	Marker_ JPG13_size_incorrect_num	Marker	Number of JPG13 (0xFFFD) in the image there size is illegal – 0 or 1	Expert	Integer
186	Marker_ JPG13_size_declared_incorrect_num	Marker	Number of JPG13 markers declare incorrect size found in the image.	Expert	Integer
187	Marker_SOF0_size_max	Marker	Size of SOF0 (0xFFC0) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
188	Marker_ SOF0_size_incorrect_num	Marker	Number of SOF0 (0xFFC0) in the image there size is illegal – 0 or 1	Expert	Integer
189	Marker_ SOF0_size_declared_incorrect_num	Marker	Number of SOF0 markers declare incorrect size found in the image.	Expert	Integer
190	Marker_SOF1_size_max	Marker	Size of SOF1 (0xFFC1) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
191	Marker_ SOF1_size_incorrect_num	Marker	Number of SOF1 (0xFFC1) in the image there size is illegal – 0 or 1	Expert	Integer
192	Marker_ SOF1_size_declared_incorrect_num	Marker	Number of SOF1 markers declare incorrect size found in the image.	Expert	Integer
193	Marker_SOF2_size_max	Marker	Size of SOF2 (0xFFC2) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
194	Marker_ SOF2_size_incorrect_num	Marker	Number of SOF2 (0xFFC2) in the image there size is illegal – 0 or 1	Expert	Integer
195	Marker_ SOF2_size_declared_incorrect_num	Marker	Number of SOF2 markers declare incorrect size found in the image.	Expert	Integer
196	Marker_SOF3_size_max	Marker	Size of SOF3 (0xFFC3) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer

197	Marker_ SOF3_size_incorrect_num	Marker	Number of SOF3 (0xFFC3) in the image there size is illegal – 0 or 1	Expert	Integer
198	Marker_ SOF3_size_declared_incorrect_num	Marker	Number of SOF3 markers declare incorrect size found in the image.	Expert	Integer
199	Marker_SOF5_size_max	Marker	Size of SOF5 (0xFFC5) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
200	Marker_ SOF5_size_incorrect_num	Marker	Number of SOF5 (0xFFC5) in the image there size is illegal – 0 or 1	Expert	Integer
201	Marker_ SOF5_size_declared_incorrect_num	Marker	Number of SOF5 markers declare incorrect size found in the image.	Expert	Integer
202	Marker_SOF6_size_max	Marker	Size of SOF6 (0xFFC6) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
203	Marker_ SOF6_size_incorrect_num	Marker	Number of SOF6 (0xFFC6) in the image there size is illegal – 0 or 1	Expert	Integer
204	Marker_ SOF6_size_declared_incorrect_num	Marker	Number of SOF6 markers declare incorrect size found in the image.	Expert	Integer
205	Marker_SOF7_size_max	Marker	Size of SOF7 (0xFFC7) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
206	Marker_ SOF7_size_incorrect_num	Marker	Number of SOF7 (0xFFC7) in the image there size is illegal – 0 or 1	Expert	Integer
207	Marker_ SOF7_size_declared_incorrect_num	Marker	Number of SOF7 markers declare incorrect size found in the image.	Expert	Integer
208	Marker_SOF9_size_max	Marker	Size of SOF9 (0xFFC9) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
209	Marker_ SOF9_size_incorrect_num	Marker	Number of SOF9 (0xFFC9) in the image there size is illegal – 0 or 1	Expert	Integer
210	Marker_ SOF9_size_declared_incorrect_num	Marker	Number of SOF9 markers declare incorrect size found in the image.	Expert	Integer
211	Marker_SOF10_size_max	Marker	Size of SOF10 (0xFFCA) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
212	Marker_ SOF10_size_incorrect_num	Marker	Number of SOF10 (0xFFCA) in the image there size is illegal – 0 or 1	Expert	Integer

213	Marker_ SOF10_size_declared_incorrect_num	Marker	Number of SOF10 markers declare incorrect size found in the image.	Expert	Integer
214	Marker_SOF11_size_max	Marker	Size of SOF11 (0xFFCB) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
215	Marker_ SOF11_size_incorrect_num	Marker	Number of SOF11 (0xFFCB) in the image there size is illegal – 0 or 1	Expert	Integer
216	Marker_ SOF11_size_declared_incorrect_num	Marker	Number of SOF11 markers declare incorrect size found in the image.	Expert	Integer
217	Marker_SOF13_size_max	Marker	Size of SOF13 (0xFFCD) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
218	Marker_ SOF13_size_incorrect_num	Marker	Number of SOF13 (0xFFCD) in the image there size is illegal – 0 or 1	Expert	Integer
219	Marker_ SOF13_size_declared_incorrect_num	Marker	Number of SOF13 markers declare incorrect size found in the image.	Expert	Integer
220	Marker_SOF14_size_max	Marker	Size of SOF14 (0xFFCE) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
221	Marker_ SOF14_size_incorrect_num	Marker	Number of SOF14 (0xFFCE) in the image there size is illegal – 0 or 1	Expert	Integer
222	Marker_ SOF14_size_declared_incorrect_num	Marker	Number of SOF14 markers declare incorrect size found in the image.	Expert	Integer
223	Marker_SOF15_size_max	Marker	Size of SOF15 (0xFFCF) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
224	Marker_ SOF15_size_incorrect_num	Marker	Number of SOF15 (0xFFCF) in the image there size is illegal – 0 or 1	Expert	Integer
225	Marker_ SOF15_size_declared_incorrect_num	Marker	Number of SOF15 markers declare incorrect size found in the image.	Expert	Integer
226	Marker_SOS_size_max	Marker	Size of SOS (0xFFDA) in the image [2 bytes after the marker]. If there are more than one – represents the max size.	Expert	Integer
227	Marker_ SOS_size_incorrect_num	Marker	Number of SOS (0xFFDA) in the image there size is illegal – 0 or 1	Expert	Integer
228	Marker_ SOS_size_declared_incorrect_num	Marker	Number of SOS markers declare incorrect size found in the image.	Expert	Integer

229	Marker_APP_major_revision_incorrect_num	Marker	Number of APP major revision incorrect value. Value must be always 1.	Expert	Integer
230	Marker_APP_minor_revision_incorrect_num	Marker	Number of APP minor revision incorrect value. Value must be 0, 1 or 2.	Expert	Integer
231	Marker_APP_unit_density_incorrect_num	Marker	Number of APP unit density incorrect value. Value must be 0x0000, 0x0001 or 0x0002.	Expert	Integer
232	Marker_APP_x_density_incorrect_num	Marker	Number of APP x density incorrect value. Value must be other than 0.	Expert	Integer
233	Marker_APP_y_density_incorrect_num	Marker	Number of APP y density incorrect value. Value must be other than 0.	Expert	Integer

2. *Feature Selection*

The features we constructed for the Knowledge-Based method are based on the understanding of former attacks and how can one exploit the JPEG file specification for construction of a malicious file; therefore our features are very general and are not based on a specific case of a malicious JPEG file and it is safe to say that using all the features without a feature selection process is safe and will not cause overfitting.

3. *Feature Representation*

In contrast to the Structural feature extraction method, we used only the count method (Sums the number of appearances of a feature in the file) for representation of the different features.

At the end of the process, we hold one dataset, containing 233 features and the number of instances of each feature in each file in our collection. We applied several machine learning classifiers on the dataset for evaluation of our proposed method for feature extraction.

C. Machine Learning Algorithms

We employed nine commonly used machine learning classification algorithms on the datasets created. We chose the following classifiers: *J48*, *Random Forest* (RF), *Naïve Bayes* (NB), *Bayesian Network* (BN), *Logistic Regression* (LR), *LogitBoost* (LB), *Sequential Minimal Optimization (SMO), Bagging* and *AdaBoost* (AB). Every different method is representing different classifier category: classification trees (*J48, RF)*, Bayesian classifiers *(NB, BN)*, function based classifiers *(LR*, *SMO)*, and meta classifiers *(LB, Bagging, AB)*.

We applied all of the above mentioned machine learning classifiers using the *Weka* data mining software with *WEKA's* default configuration. We used the following configurations of classifiers: *J48* with a confidence factor (used for pruning) of 0.25 and a minimum of two instances per leaf; *Random Forest* classifier twice, once with 100 trees and again with 500 trees, with no limitation regarding the depth of the trees; *LogitBoost* classifier based on the *DecisionStump* classifier; *SMO* classifier with a polynomial kernel; *Bagging* classifier based on the *REPTree* classifier; and *AdaBoost* based on the *DecisionStump* classifier.

VI. Evaluation

A. Research Questions

In order to evaluate the effectiveness of the proposed methods presented earlier (structural feature extraction methodology and knowledge-based features) for the detection of unknown malicious *.jpg images using machine learning algorithms, we first wanted to determine which configuration yields the best detection accuracy measures. The research questions are as follows:

1. *Structural Feature Extraction Method*

2. Are machine learning algorithms trained on features extracted using the structural feature extraction methodology presented earlier useful for the efficient detection of malicious JPG images?
3. Which feature extraction configuration provides the best detection results?
4. Which feature selection method provides the best detection results: *Information Gain* or *Information Gain Ratio?*
5. Which feature representation method provides the best detection results: *Binary*, *Count*, *TF* or *TFIDF?*
6. Which top-features dataset provides the best detection results: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1,000, or 2,000?
7. Which classifier provides the best detection results: *Naïve Bayes*, *Bayes Network*, *J48*, *Random Forest*, *Logistic Regression*, *LogitBoost*, *SMO*, *Bagging,* or *AdaBoost?*
8. Which configuration of feature extraction, feature selection, feature representation, top-feature selection, and classifier provides the best detection results?

2. *Knowledge-Based Feature Extraction Method*

1. Are machine learning algorithms trained on the knowledge-based features presented earlier useful for the efficient detection of malicious JPG images?
2. Which classifier provides the best detection results: *Naïve Bayes*, *Bayes Network*, *J48*, *Random Forest*, *Logistic Regression*, *LogitBoost*, *SMO*, *Bagging,* or *AdaBoost?*
3. Which top 10 features found to be the mostly contribute to the discrimination between malicious and benign JPG images?

3. *Both Knowledge-Based Feature and Structural Feature Extraction Method*

1. Does the best configuration structural feature extraction provide better detection results than existing anti-virus engines?

B. *Evaluation Measures*

For evaluation purposes, we measured the AUC (area under the receiver operating characteristic (ROC) curve). The ROC curve is created by plotting the TPR (true positive rate) against the FPR (false positive rate) at various threshold settings. In order to achieve high AUC, high TPR and low FPR are needed to check at each threshold. The AUC is a preferred measure (over the accuracy measure) for comparing machine learning algorithms applied on datasets (balanced or imbalanced binary or multiclass) as suggested by [HuLi05].

C. *Experimental Design*

To answering the Research questions that we wrote above, we designed three Experiments. Every experiment will help us to collect the relevant information for the research questions.

1. *Experiment 1 – Finding the Best Overall Configuration for Structural*

The first experiment relates to research question 1-7 in the Structural Feature Extraction Method category. The experiment will evaluate the results achieved by applying each classifier of the nine on 1280 datasets and compare the results between them. The results will be related to the average AUC achieved with all the different configurations (feature extraction, feature selection, feature representation, top-feature selection and classifier).

2. *Experiment 2 – Detection results of knowledge-based*

The second experiment relates to research question 1-2 in the Knowledge-Based Feature Extraction Method category. The experiment will evaluate the results achieved by applying each classifier of the nine on the dataset and compare the AUC results between them.

3. *Experiment 3 – Comparison with Anti-Virus Engines*

The third experiment relates to the research question in the Both Structural and Knowledge-Based feature extraction Methods category. The experiment will compare the TPR provided by the best configuration of the Structural and Knowledge-Based feature extraction methods with the different Anti-Virus softwares. We used VirusTotal’s online web service which provided analysis of 64 commonly used Anti-Virus engines, to analyze the malicious files in our collection. We analyzed the report produced by VirusTotal to compute the TPR for each Anti-Virus engine.

D. Results

1. Results of Experiment 1

Experiment number 1 examined the performance of the structural feature extraction method. It answers the seven research question on the Structural Feature Extraction Method category.

The first research question checks if the structural feature extraction method is a good way of detection malicious files. The answer for this question is yes and we will demonstrate it through the other questions.

We examined the five parts of the process (Extraction, Selection, Representation, Top features and Classifier) independently and together. In Figure 15 we can observe the average AUC of every feature extraction configuration. The best configuration is FFT which provided 0.9262 average AUC. From those results, we can say that the best configuration is the Single Joint.

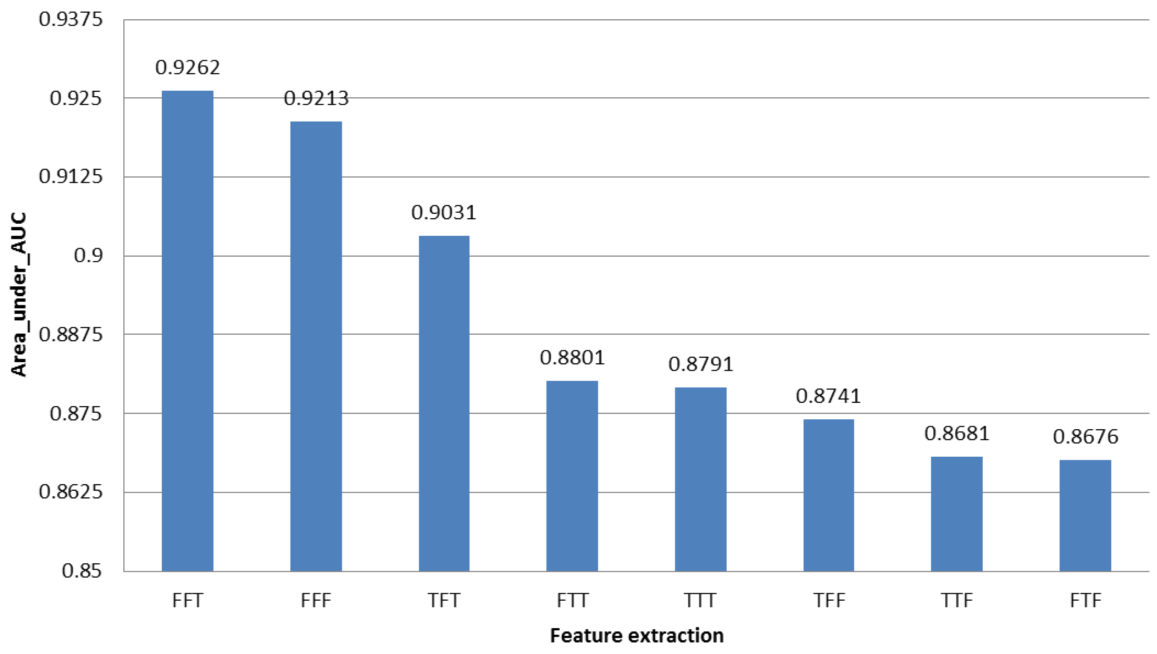


Figure 15. Comparison between the different configurations – Structural Feature Extraction method

In Figure 16 we can observe the average AUC of every feature selection method. The best feature selection method is Information Gain which provided 0.9255 average AUC.

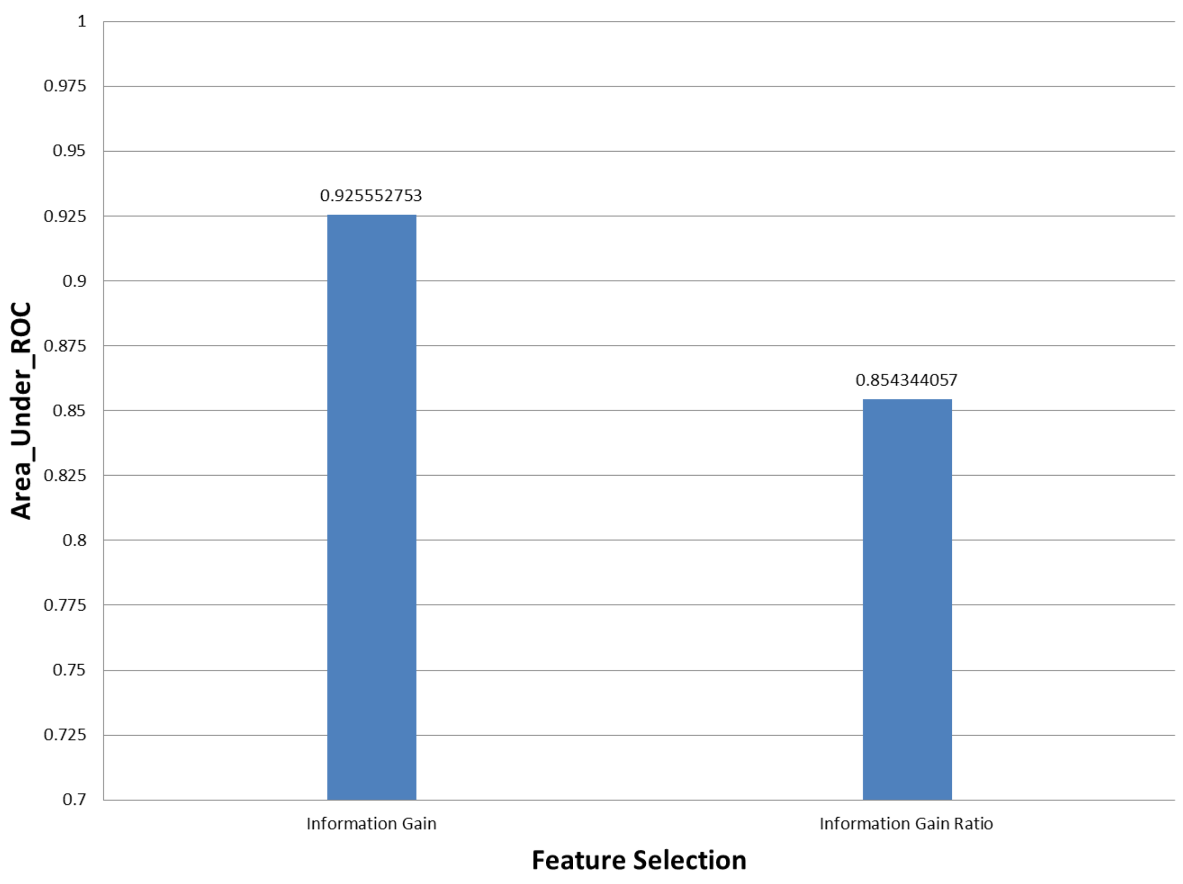


Figure 16. Comparison between the different selections – Structural Feature Extraction method

In Figure 17 we can observe the average AUC of every feature representation method. The best representation method is Binary which provided 0.9451 average AUC.

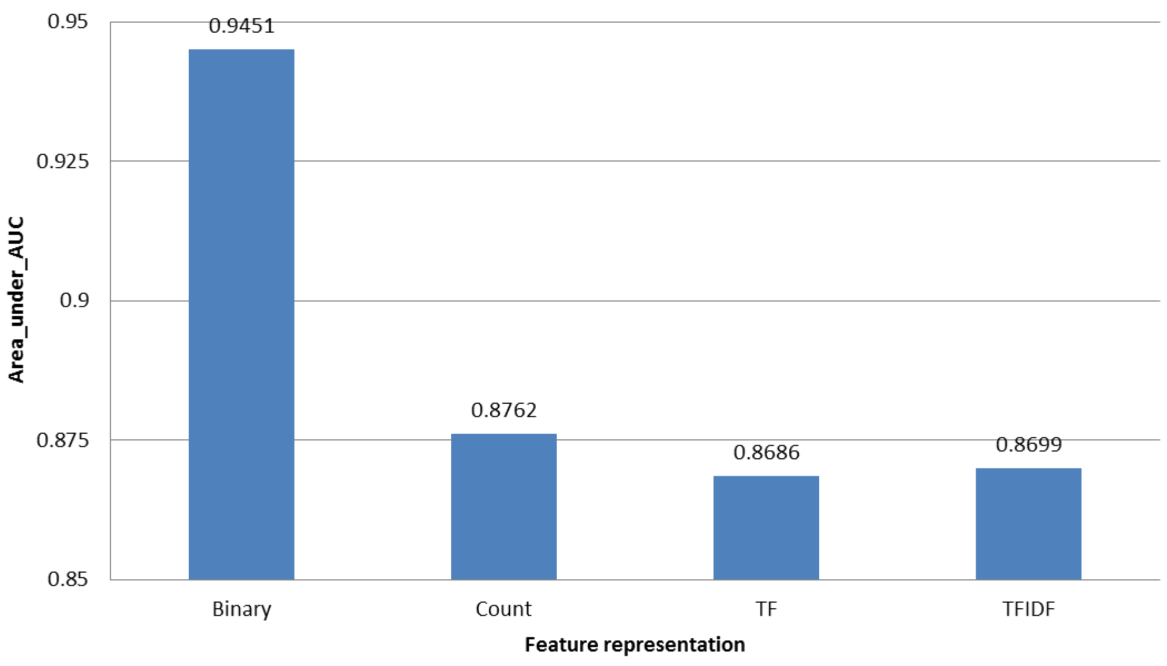


Figure 17. Comparison between the different representations – Structural Feature Extraction method

In Figure 18 we can observe the average AUC of every number of top features. The best top features is 100 which provided 0.8967 average AUC.

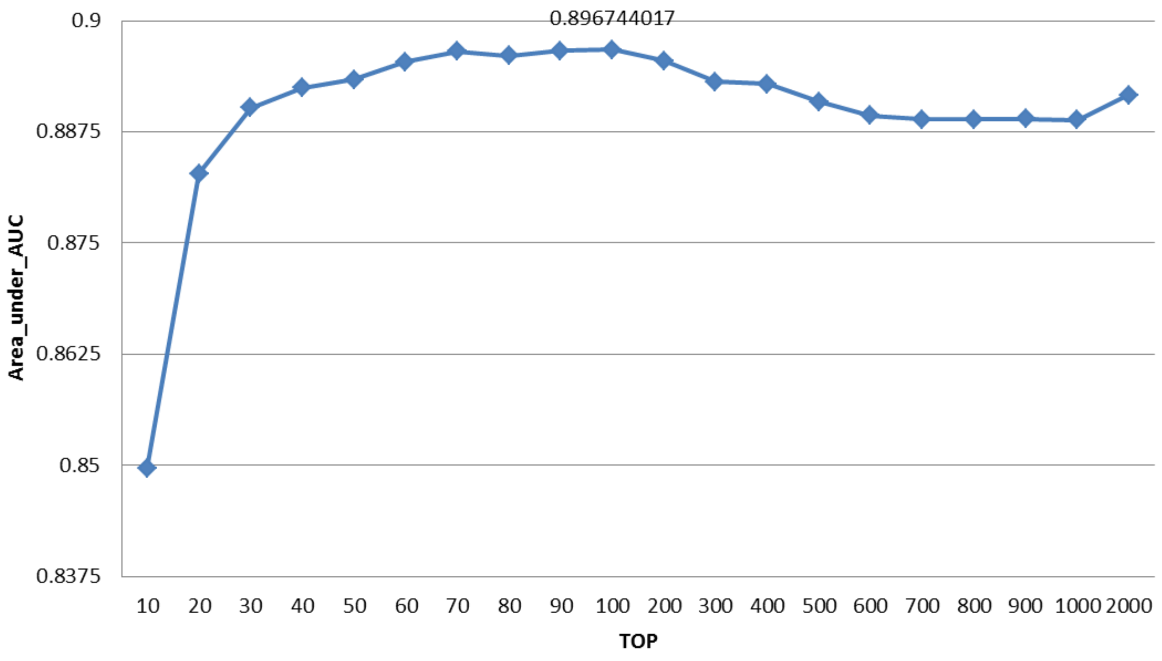


Figure 18. Comparison between the different Top Features – Structural Feature Extraction method

In Figure 19 we can observe the average AUC of every classifier. The best classifier is LogitBoost which provided 0.951 average AUC.

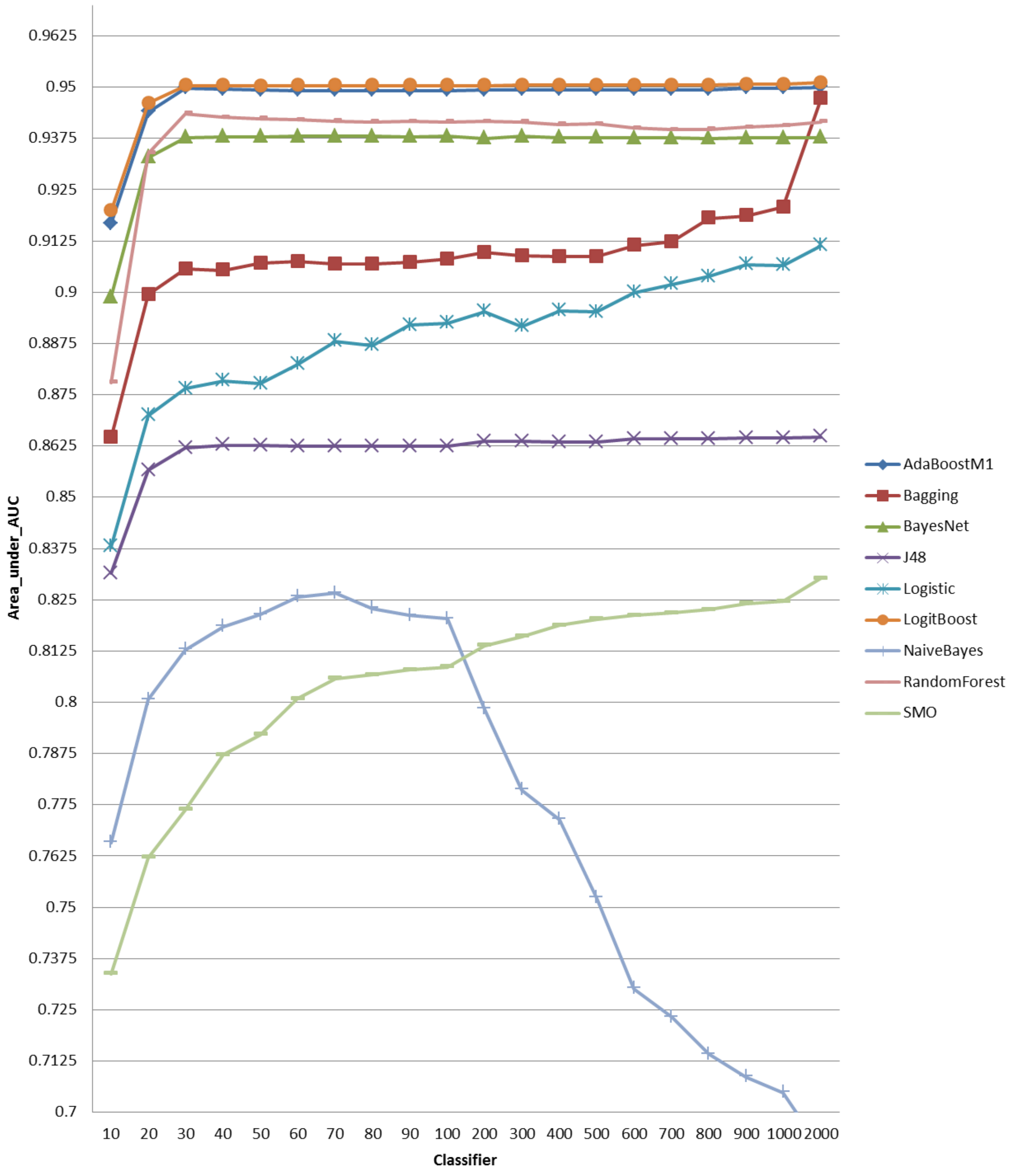


Figure 19. Comparison between the different classifiers – Structural Feature Extraction method

To answer the sixth research question, determining the best configuration, we examined the configuration with the highest AUC. We searched for the most prevalent condition for each factor. In **Error! Reference source not found.** we can observe TPR, FPR, AUC and F-Measure of the best combination (Configuration, Selection, Representation, Top features and Classifier). The best combination is TFT extraction, Count representation, Info Gain selection, 2000 Top features and Random Forest classifier which provided 0.9084 TPR, 0.0028 FPR and AUC 0.9716. It's important to mention that this is the combination that provided average best performances in a 10-fold cross-validation format. In conclusion, the results provided by machine learning algorithms trained on the structural feature extraction method are useful for efficient detection of malicious JPEG files.

Feature Extraction	Feature Representation	Feature Selection	Top Features	Classifier	TPR	FPR	AUC	F-Measure
TFT	Count	Info Gain	2000	Random Forest	0.9084	0.0028	0.9716	0.9407

Table 12. Results of the best combination (Configuration, Selection, Representation and Classifier) - Structural Feature Extraction method

2. *Results of Experiment 2*

Experiment number 2 examined the performance of the knowledge-based feature extraction method. It answers the two research question on the Knowledge-Based Feature Extraction Method category.

The first research question checks if the knowledge-based feature extraction method is a good way of detection malicious files. The answer for this question is absolutely yes.

In Figure 20. Knowledge-Based Feature Extraction method AUC**Error! Reference source not found.** we can observe the average accuracy ratio of every classifier. The best classifier is LogitBoost which provided 0.9995 average AUC. The results indicate that Knowledge-Based feature extraction method is a good detection method which provides high AUC values.

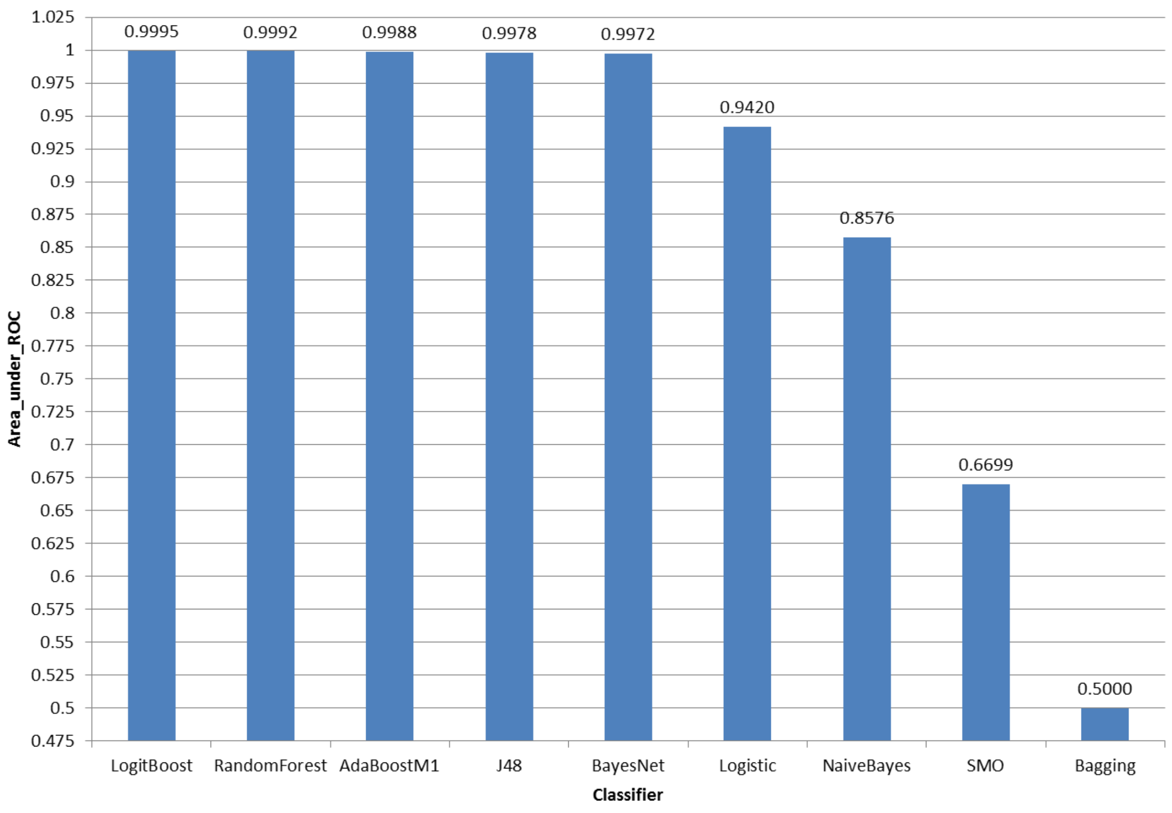


Figure 20. Knowledge-Based Feature Extraction method AUC

3. *Results of Experiment 3*

Experiment number 3 compares the true positive rate (TPR) between the best configurations found in the previous experiment of the structural feature extraction method, knowledge-based feature extraction method with the Anti-Virus softwares detection TPR. This will show the best way for detection malicious JPEG files. In Figure 21 we can observe the comparison between our two methods and familiar Anti-Virus softwares. The method which provides the best results is Knowledge-Based feature extraction with 0.9787 TPR. This method surpasses all the best Anti-Virus softwares and the Structural method. In addition, we can see that both of Structural and Knowledge-Based detection methods capabilities surpass all the Anti-Virus softwares in the task of detecting malicious JPG files.

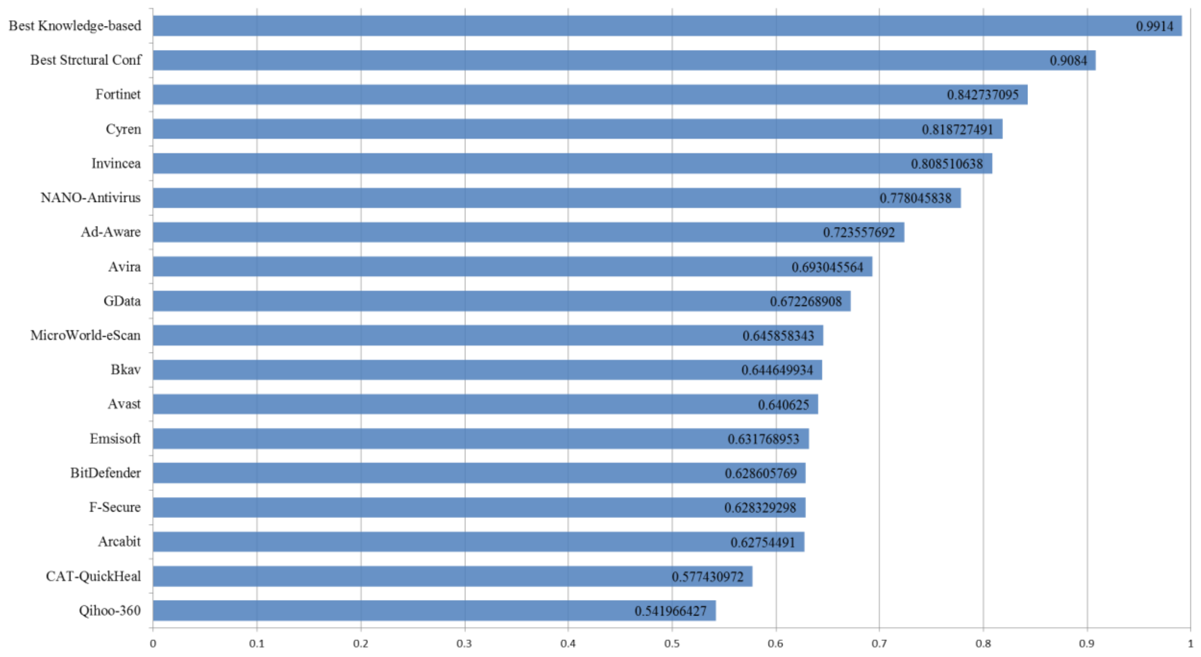


Figure 21. Comparison between accuracy of Structural, Knowledge-Based and Anti-Virus Software.

VII. DISCUSSION AND CONCLUSIONS

To sum up, we conducted two main theories for detection malicious JPG images. The first one is using a structural feature extraction method and elaborate it. This method was examined in the past (A. Cohen, N. Nissim, L. Rokach, and Y. Elovici, 2016) by creating a tree of hierarchy strings from Top to Bottom. We conducted creation of strings both Top to Bottom, Bottom to Top and single joint. In the experiments, we found out that the Single Joint is a great way for detection malicious files and the Bottom to Top is not. The second theory was learning former attacks, known vulnerabilities and the structure of the file (How it supposed to be built). This method provided a very high detection ratio (approximately 99% correct detection). This method is a new way of detection that surpass even familiar Anti-Virus softwares. It's not depending on specific dataset but a former knowledge that we summed up to features. Thus, we aren't getting overfitting. We believe this method could detect Zero-day attacks and of course known attacks with a high accuracy ratio.

A. Limitations

Our project was revealed a new method for detection both known and zero days attacks that surpass even known Anti-Virus softwares. Although, we have some limitations to our work. First of all, the comparison between the Anti-Virus softwares and our methods, we didn't compare our methods with 0 FPR like the Anti-Virus softwares. Secondly, the Structural feature extraction method was used files from the dataset for creation the best features. This situation can lead to overfitting (We tried to handle it by using 10-cross fold validation).

B. Future Work

In our project we examined and displayed a new ways of creation of features for feature extraction (Top to Bottom, Bottom to Top and Singal Joint). More over, we checked the use of former attacks and understanding the file structure for creation of strong and indicative features that supposed to detect known attacks and even zero-day attacks.

Future work supposed to check this methodology on different kind of file like PNG, GIF etc.

VIII. ACKNOWLEDGMENTS

We would like to thank the Malware Lab at the Cyber Security Research Center (CSRC) at Ben-Gurion Univiversity of the Negev, Israel, for supporting this research. Many thanks also to *VirusTotal* for granting us complimentary access to their private services.

IX. MEETINGS TABLE

Date	Participated	Subject	Summary
19.06.16	Aviad Alex Liad	Introduce the project	• Display the project
17.08.16	Aviad Alex Liad	First work meeting	• Start collecting articles • Start readong about RTF and JPG
02.11.16	Nir Nissim Aviad Alex Liad	Second work meeting	• Understand the RTF and JPEG structures • Finding attacks, Vulnerabilities and existing detection Tools
05.12.16	Aviad Alex Liad	Third work meeting	• Presenting the RTF and JPEG structures • Presenting attacks, Vulnerabilities and existing detection Tools • Literature Review
09.01.17	Aviad Alex Liad	Fourth work meeting	• Fine-tuning on Reports • Knowledge-based features and Structural features
24.01.17	Aviad Liad	Fifth work meeting	• Define the programming methods • Fine-tuning on Reports
24.04.17	Aviad Alex Liad	Sixth work meeting	• Finishing the programming phase
24.04.17	Aviad Alex Liad	Seventh work meeting	• Working with Weka • Reports fine-tuning
08.05.17	Aviad Alex Liad	Eighth work meeting	• Results of experiments • Reports fine-tuning
15.05.17	Aviad Alex Liad	Ninth work meeting	• Poster • Results of experiments • Reports fine-tuning
05.06.17	Aviad Alex	Tenth work meeting	• Poster • Fine-tuning on Reports

	Liad		

Table 13. Meetings Table

X. REFERENCES

[ACKS00] ABOU-ASSALEH, TONY ; CERCONE, NICK ; KEˇ, VLADO ; SWEIDAN, RAY: N-gram-based Detection of New Malicious Code Bd. 1, Nr. 1, S. 2–3

[ANWN07] ABU-NIMEH, SAEED ; NAPPA, DARIO ; WANG, XINLEI ; NAIR, SUKU: A Comparison of Machine Learning Techniques for Phishing Detection (2007)

[Chri00] CHRISTODORESCU, MIHAI: Static Analysis of Executables to Detect Malicious Patterns ∗

[CNRE16] COHEN, AVIAD ; NISSIM, NIR ; ROKACH, LIOR ; ELOVICI, YUVAL: SFEM : Structural feature extraction methodology for the detection of malicious office documents using machine learning methods. In: *Expert Systems With Applications* Bd. 63, Elsevier Ltd (2016), S. 324–343

[FCKV00] FORD, SEAN ; COVA, MARCO ; KRUEGEL, CHRISTOPHER ; VIGNA, GIOVANNI: Analyzing and Detecting Malicious Flash Advertisements

[HaVi00] HALLARAKER, OYSTEIN ; VIGNA, GIOVANNI: Detecting Malicious JavaScript Code in Mozilla

[HuLi05] HUANG, JIN ; LING, CHARLES X.: Using AUC and accuracy in evaluating learning algorithms. In: *IEEE Transactions on Knowledge and Data Engineering* Bd. 17 (2005), Nr. 3, S. 299–310 — ISBN 1041-4347

[JoKK00] JOVANOVIC, NENAD ; KRUEGEL, CHRISTOPHER ; KIRDA, ENGIN: Pixy : A Static Analysis Tool for Detecting Web Application Vulnerabilities (Technical Report)

[MaGi00] MAIORCA, DAVIDE ; GIACINTO, GIORGIO: Looking at the Bag is not Enough to Find the Bomb : An Evasion of Structural Methods for Malicious PDF Files Detection — ISBN 9781450317672

[NCGE14] NISSIM, NIR ; COHEN, AVIAD ; GLEZER, CHANAN ; ELOVICI, YUVAL: ScienceDirect Detection of malicious PDF files and directions for enhancements : A state-of-the art survey. In: *Computers & Security* Bd. 48, Elsevier Ltd (2014), S. 246–266

[Otsu15] OTSUBO, YUHEI: O-checker : Detection of Malicious Documents through

Deviation from File Format Specifications (2015)

[PeMi93] PENNEBAKER, WILLIAM B ; MITCHELL, JOAN L: *JPEG still image data compression standard*. Bd. 34, 1993 — ISBN 0442012721

[PrST16] PRIYANKA, RAMA ; SAHOO, P K ; TECH, STUDENT M: Scanning Tool For Identification of Image With Malware Bd. 4 (2016), Nr. 1, S. 170–175

[Serv08] SERVER, WINDOWS: Rich Text Format (RTF) Specification Rich Text Format (RTF) Specification (2008)

[SPDB00] SANTOS, IGOR ; PENYA, YOSEBA K ; DEVESA, JAIME ; BRINGAS, PABLO G: N-GRAMS-BASED FILE SIGNATURES FOR MALWARE DETECTION

[ThSc13] THESIS, BACHELOR ; SCIENCE, COMPUTER: Malicious PDF Document Analysis (2013), S. 1–20

[WeKa89] WEISS, SHOLOM M ; KAPOULEAS, IOANNIS: An Empirical Comparison of Pattern Recognition , Neural Nets , and Machine Learning Classification Methods (1989), S. 781–787