Detecting Stegomalware: Malicious Image Steganography and Its Intrusion in Windows

Vinita Verma, Sunil K. Muttoo, and V. B. Singh

1 Introduction

Steganography, a technique to hide data within digital media, has primarily been used to communicate secret data, embed watermarks for copyright protection, etc. This practice, however, has trended into hiding the malware within digital media to evade detection. Such malware is known as stegomalware. Specifically, the popularity of seemingly innocuous digital images shows a high potential for the malicious use of image steganography. Images across the internet, social media, email attachments, or resources in the applications can be exploited to hide malicious code, configuration data, or URL to retrieve the code or other components from an external server [1]. Such images though stay undetected by intrusion detection systems or static analysis that typically lack analyzing the steganography in images and thus anything suspicious within images. Static analysis analyzes opcodes, control flow graphs, n-grams, etc., within the code without executing it. However, it fails to identify malicious components other than the code such as images and is thwarted by code obfuscation. Dynamic analysis unaffected by obfuscation executes the code to trace malign actions. This analysis is, however, un-scalable for large-scale detection being time- and resource-intensive. Alternatively, signature-based detection fails to detect unknown samples which contain new malicious patterns or signatures for every attack. These factors lead malware developers to a more secure obfuscation technique, a lucrative method of steganography, specifically image for hosting malware.

The malicious role of steganography is discussed in the McAfee 2017 report [2]. The years 2011–2017 witnessed steganographic threats [3]. According to the

V. Verma (✉) · S. K. Muttoo
Department of Computer Science, University of Delhi, Delhi, India

V. B. Singh
Department of Computer Science, Delhi College of Arts and Commerce, University of Delhi, Delhi, India

U. P. Rao et al. (eds.), *Security, Privacy and Data Analytics*, Lecture Notes in Electrical Engineering 848, https://doi.org/10.1007/978-981-16-9089-1_9

Criminal Use of Information Hiding (CUING) [4], which is an initiative launched in cooperation with Europol's European Cybercrime Centre, stegomalware with respect to the malware discovered between 2011 and 2016 hiked from 12% to 24%. A survey of node capture attack in wireless sensor networks [36] and an optimization technique [37] is presented to escalate its attacking efficiency.

The literature has discussed some methods to detect malicious image steganography. A method was proposed to find a URL [5] hidden in LSBs for all kinds of images. However, only the LSB hiding technique was considered. Anomalous data appended to GIF images [6] was detected based on locating the end of the file. DCT-based techniques [7], smart threshold, and anomaly correction were proposed against cyberattacks that exploit images and video streams, applicable for JPEG images and H.264 I-Frames. It resulted in 80% protection against cyberattacks with 25.74 dB PSNR for an aggressive attack configuration. Data gathering or kernel tracing-based [8] stegomalware detection was proposed. Its future scope, however, aims at a more programmatic approach for tracing execution patterns for stegomalware or specific covert channels. A study was performed on favicons [9], the icons associated with webpages or websites, using state-of-the-art steganalysis and features exploiting flat areas in them. It detected Vawtrak malware's steganography in favicons though steganalysis was not found the same as in natural images. It is discussed an image security technique using cryptography and steganography [38].

Many works have been discussed related to stegomalware for Android OS relative to Windows. Stegomalware for smartphones [10] was demonstrated via an app with malicious executable components within its assets. However, preliminary results of detection were not conclusive but revealed a considerable amount of data hidden in the app assets. A malicious Android application was hidden inside JPEG/PNG images [11] followed by storing the images in resources of another application to show the ease of trivial hiding methods in evading anti-malware. Our work differs from this approach in a way that the authors [11] indeed created malware by hiding malicious applications inside images while our work has used a real-world dataset of malicious images which are not just limited to containing an application but URLs, deviations from standard image format, statements comprising malicious function, etc. Moreover, no such work has been demonstrated for the Windows platform to the best of our knowledge. Steganography was used as one of the threat models [12] to implement a malign Android app to pose attacks. As follows in observation, the image chunks failing to meet file format specification are ignored by the picture viewer, thereby used to insert malicious codes. A method to detect obfuscated malware components within smartphone apps [13] was considered for stegomalware detection. It analyzes behavioral differences between the original app and the modified version with faults injected. However, the method uses a dynamic approach.

It has, therefore, been observed a lack of effective methods when it comes to locating malicious data within images. Also, relatively no such work has been found in the literature that studies or demonstrates stegomalware in Windows applications. However, this paper has addressed both issues. The remainder of the paper is structured as follows. Section 2 reviews related work. Section 3 describes the JPEG file format. Section 4 provides a methodology for the proposed tool and Sect. 5 discusses

the experiment. The results are analyzed in Sect. 6. Section 7 examines the detection of stegomalware within Windows applications by available antiviruses. Section 8 concludes the paper.

2 Related Work

The first machine learning-based method to detect malicious JPEG images [14] used ten features derived from the JPEG file structure. It resulted in an FPR of 0.004, TPR of 0.951, and AUC of 99.7%. However, this method contains no mechanism to give information on the hidden malicious content or its location within images. Moreover, no analysis has been performed for the images with non-malicious data hidden which could also impact the file structure. Steganalysis based on an Artificial Immune System (AIS) [15] was proposed to detect JPEG images modified with specific steganography tools. Using haar wavelet, horizontal coefficients attained the best steganogram detection rate of 94.33% while vertical and diagonal coefficients reached an average detection rate of 85.71%. Different image entropies-based [16] a method was proposed to detect stegobot, a social network security threat that uses images on social networks to spread the malware. It attained almost 80% accuracy for the images embedded using different JPEG hiding techniques. The method needs further computations for the network-level defense of botnets with scalability an issue in the social networks. A framework identifying malicious JPEG images [17] over social networking sites consisted of three phases: (a) use of available steganalysis methods to find steganography artifacts (b) extraction of embedded data to identify file header (c) uploading that data to VirusTotal [18] to confirm the results. The method, however, has not used any real-world datasets and the hidden data may not necessarily be an application with a header part, failing the detection. Fridrich et al. [19] presented an overview of feature-based steganalysis for JPEG images and its implications for the future design of stegosystems. A method was proposed to distinguish legitimate JPEG operations [20] like compression from malicious ones. However, 'malicious' in the context of the work does not refer to an image containing malicious data, but rather an unauthentic and manipulated image.

Many steganographic threats have been observed in the wild. In July 2013, a backdoor [21] was found that compromised a site. It contained PHP functions within the JPEG header to read the header part and execute the content. Saumil Shah [22] at the 2015 Black Hat conference introduced the term 'stegosploit' referring to an image-based exploit. The exploits are embedded into JPG and PNG images with HTML and JavaScript code, producing an HTML+ image polyglot which seems an innocuous image but triggered in the victim's browser. Facebook Messenger [23] was reported in November 2016 for using JPEG images to spread Locky ransomware while in August 2017 [24], JPEG images spread the SyncCrypt ransomware. In December 2018, Trend Micro [25] reported the use of memes (JPEG) on Twitter to communicate with the malware. A LokiBot malware [26] was noted in August 2019 for an up-gradation hiding its source code in images. In December 2019, a cybersecurity

company reported the use of JPEG of Taylor Swift [27] to hide MyKings crypto mining botnet.

Given the rising cyber threats via JPEG images, a comprehensive detection method is needed. The existing works on JPEG images have not exactly focused on revealing the hidden malicious data but rather finding artifacts left behind by the hiding tools or on feature-based detection which lacks this functionality to reveal the hidden content. This paper though alongside classification attempts to locate malicious content in JPEG images and produce that data as the output.

3 JPEG Format

JPEG stands for Joint Photographic Experts Group, named after a committee that created the JPEG standard in 1992. This file format is used by image capturing devices such as digital cameras and to store and transmit photographic images on the web. It is a widely used image format primarily due to its lossy compression ability. JPEG images have a file extension of .jpg/.jpeg. A JPEG image consists of a sequence of segments where each segment begins with a two-byte indicator called a marker. Every marker starts with the 0xFF byte (hexadecimal notation) followed by another byte that indicates a kind of data the respective segment holds. The markers are followed by the two bytes indicating the size of the segment-specific data that follows including two bytes for itself. Few segments though don't contain any payload and consist of just two marker bytes. A JPEG image starts with 0xFFD8 marker bytes (SOI_Start of Image), followed by the application-specific 0xFFEn markers (APPn) holding metadata where n = 0... F. The 0xFFDA marker (SOS_Start of Scan) contains the compressed image data where restart markers 0xFFD0 through 0xFFD7 inserted at regular intervals separate independent chunks of the data to allow parallel decoding. The 0xFFD9 or EOI marker denotes the end of the image. Table 1 provides JPEG-specific markers with their corresponding bytes and description.

4 Tool Proposed

We have created a tool in python (a python script) to detect malicious JPEG images. The tool is provided an input—a JPEG image. It reads the image, scanning and locating specific marker bytes and interpreting the data in respective segments to find any malicious content or deviation from the JPEG format. Based on this, it classifies the image into malicious or non-malicious. Besides classification, the tool contains the functionality of revealing the malign data found within the image along with its location as part of the output. Concerning the detection of stegomalware with respect to images, this functionality to the best of our knowledge has not been found in the available literature. It would enhance insight into the kind of malicious

Table 1 JPEG image markers

Marker	Bytes	Data	Description
SOI	0xFF 0xD8	None	Start of image
APPn	0xFF 0xEn, n = 0... F	Variable size	Application specific
COM	0xFF 0xFE	Variable size	Comment
SOF0	0xFF 0xC0	Variable size	Start of frame (Baseline JPEG)
SOF2	0xFF 0xC2	Variable size	Start of frame (Progressive JPEG)
DHT	0xFF 0xC4	Variable size	Define huffman table(s)
DQT	0xFF 0xDB	Variable size	Define quantization table(s)
RSTn	0xFF 0xDn, n = 0... 7	None	Restart
DRI	0xFF 0xDD	4 bytes	Define restart interval
SOS	0xFF 0xDA	Variable size	Start of scan
EOI	0xFF 0xD9	None	End of image

content hidden within images. The proposed tool flags an image as malicious under either of the following scenarios and non-malicious in case none of the scenarios are detected, as also illustrated in a flowchart presented in Fig. 1.

a. The tool reads and checks the starting two bytes in an input JPEG image. If the image is not found starting with the 0xFFD8 bytes (SOI marker), the image is reported as malicious for deviating from the JPEG format. It is uncommon and suspicious for an image to start with any other bytes than the SOI marker.
b. The tool locates application-specific markers by their corresponding bytes and scans the metadata that follows to find any malicious content. The corresponding ASCII code of the bytes comprising the metadata is obtained which is searched for some suspicious keywords or strings such as 'script', 'eval', and 'iframe' using a string finding function. The 'script' keyword has been used to locate a <script> tag which specifies a JavaScript file to load. Another keyword, a JavaScript function 'eval' has been searched for which is used to evaluate or execute the arguments passed to it. The argument can be any expression or one or more JavaScript statements. The 'iframe' stands for Inline Frame. Locating this tag identifies a suspicious action of embedding another document within the current HTML document. If either of the strings is found, the image is flagged as malicious.
c. The tool locates 0xFFFE bytes and searches in the respective comment segment for the suspicious strings in the same way it does for the metadata. The image is predicted as malicious upon finding either of the strings.
d. The tool searches for the bytes 0xFFD9 indicating the end of the image. If the image is found missing these bytes (EOI marker), the image raises suspicion for not following the standard JPEG format, thereby predicted as malicious.
e. The tool on locating the EOI marker, if finds any anomalous data appended to the end of the image, i.e., finding any data after the 0xFFD9 bytes, flags the image as malicious. The maliciousness of such data is identified using the same

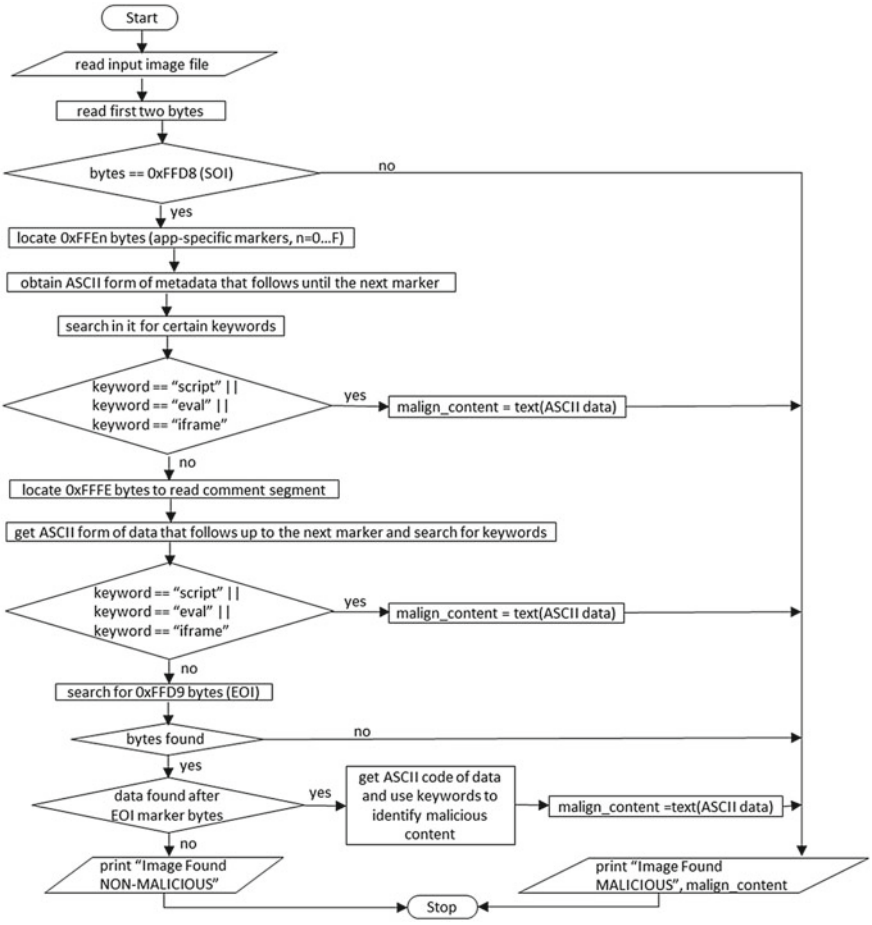

Fig. 1 Working of tool proposed

strings as used for the metadata and comment. Such trailing data is usually hard to notice for it comprises a few bytes causing a negligible change in the file size.

In cases (b), (c), and (e) as mentioned above that involve locating the hidden malign data, the tool outputs/produces the found malicious content along with its location.

5 Experiment

We have collected three types of JPEG images for the experiment: malicious, benign images hiding no data, and stego images that contain non-malicious data hidden. Though malicious images hiding malign content are also stego images, the ones

hiding non-malicious data have been referred to as the stego images in the context of this paper. Therefore, stego images can be categorized into benign images for they are harmless, not containing any malware. The purpose behind using stego images is to evaluate the effectiveness of our tool in classifying the images hiding malicious and non-malicious data. Many existing works have focused on finding steganography artifacts [9, 15, 17, 19] produced by specific algorithms. However, the presence of steganography may indicate malware but it is not sufficient enough to reach conclusions since images can use steganography for legitimate purposes as well such as copyright protection. Thus, we want to ensure that the tool proposed doesn't just rely on steganography artifacts but rather inspects the content of the hidden data for detection.

We have collected about 5,893 JPEG images, containing 2,620 malicious, 3,185 benign, and 88 stego images. Malicious images have been collected from the malware repositories [28, 29]. We have analyzed VirusTotal reports for every malicious sample to ensure their maliciousness. The size of malicious images ranges from 22 bytes to 1,358 KB. Besides, 3,185 benign images have been collected from multiple sources [29–32]. These images range from 1 KB to 13,154 KB in size. Also, we have used about 40 stego images collected from our previous work [33] which used steganography, hiding non-malicious data in JPEG images with improved capacity and imperceptibility. The paper [33] aimed to protect sensitive data in PCs via hiding such data within image resources present on the systems from any malicious attempt of stealing or tampering with data. The stego images collected consist of eight images each with different embedding rates of 0.2, 0.4, 0.6, 0.8, and 1.0, and vary in size from 32 to 76 KB. These stego images depict spatial domain steganography. Therefore, to extend the evaluation, about 48 JPEG stego images have been further generated that hide non-malicious data in the transform domain. For this, we have used the F5 steganography algorithm [34], running it via Java from the command line. The size of these images lies between 5 and 175 KB. In total, 88 (40 + 48) stego images have been collected. All the benign and stego images have been analyzed using VirusTotal to verify if the files are clean. After collecting the samples, we have run the proposed tool, the python script, via command prompt in Windows for each of the 5,893 images, providing file name as the input.

6 Results

The tool proposed has analyzed every input JPEG image into malicious and non-malicious as per the procedure mentioned in Sect. 4. The stego images have been named S1, S2, and so on for convenience. Being harmless as containing non-malicious data, the stego images have been included in the dataset of benign images to produce a collective result. Running the tool on 2,620 malicious and 3,273 (3,185 + 88) benign images, we have observed and noted the results for every image. Summarizing the results, the precision, a ratio of correctly classified positive (malicious) samples to the total samples predicted as positive, obtained is 0.99. On other

Table 2 Comparative analysis of our technique with state-of-the-art techniques

Techniques	Detection rate (%)	Output the malign data found within images
Entropy-based [16]	~80	✗
AIS [15]	94.33	✗
MalJPEG [14]	94.8	✗
Proposed tool	96.16	✓

hand, recall (sensitivity) which is a ratio of correctly classified positive samples to the total positive ones, attained is 0.91. Considering both precision and recall, the F-measure of 95.50% has been obtained and an accuracy of 96.16% using our method. Other important measures that evaluate the classification performance are the False Negative Rate and the False Positive Rate. FNR is defined as a ratio of the positive samples misclassified as negative (benign) to the total positive ones. FPR is a ratio of negative samples misclassified as positive to the total negative ones. The lower the FNR and the FPR, the higher is the performance. The tool has resulted in an FNR of 0.08 and FPR of 0.001 which are fairly low. Indeed, all the stego images have been predicted as non-malicious, indicating the effectiveness of the proposed tool in distinguishing the images hiding malicious and non-malicious data. This can be attributed to the use of certain strings by the tool for locating malicious data within images. On the other hand, misclassification of malicious samples can be attributed to the presence of malware in the locations not covered by the tool such as LSBs. However, such cases are less likely since JPEG uses a lossy compression technique which can disrupt the data hidden in LSBs. The easy and more likely JPEG hiding methods consist of embedding the data in the header, comments, or at the end of the image file. Therefore, we have focused on finding the suspicious data at these locations in JPEG images.

A comparison of our result with that of state-of-the-art techniques [14–16] is presented in Table 2. Unlike our technique, the works [14–16] have used feature-based analysis for the detection of JPEG images. The table shows that our tool has relatively obtained a better detection rate with the functionality to output the malicious content found within images.

Other than the classification results, the distribution of threats in malicious images used in this paper has been explored as shown in Fig. 2. The figure shows that the majority of malicious images, i.e., around 94% contain suspicious data appended to the end of the image file. About 3% of images contained malware in the header comprising the metadata while the ones with malicious comment segments have been found with 0.04% distribution. The images found not following the standard JPEG format in terms of SOI and EOI markers, account for around 1 and 2% of the distribution, respectively.

Another observation being illustrated in Fig. 3 is regarding the presence of suspicious strings within the malicious images used. The samples have been found containing the HTML code for malicious purposes. The figure shows that about 53% of images contain the script tag enabling malicious JavaScript and PHP files

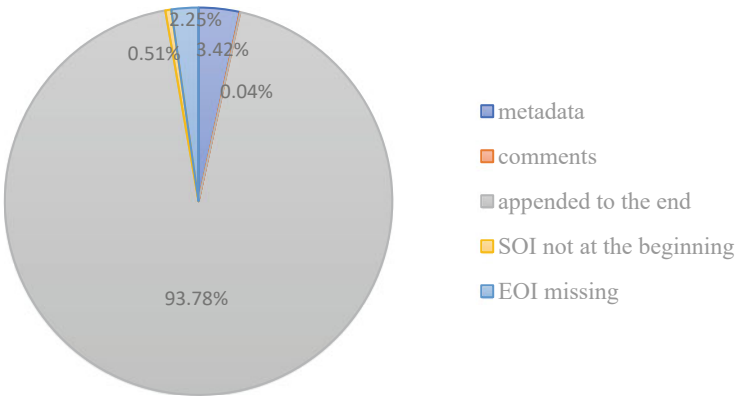

Fig. 2 Distribution of threats in malicious JPEG images

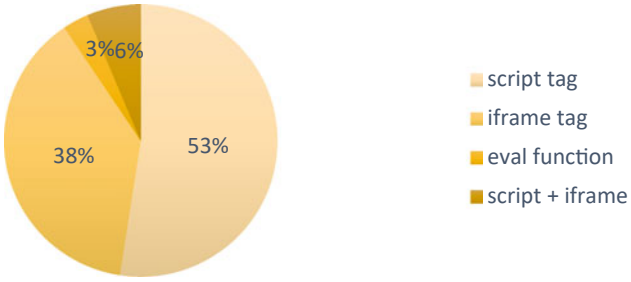

Fig. 3 Presence of suspicious strings within malicious JPEG images

and 38% comprise an iframe tag with hidden visibility while 6% contained both the tags. About 3% of images contained the JavaScript eval function for executing malicious string arguments. It could end up running malicious code on the targeted machine. An instance of the output for malicious and stego images is depicted via a screenshot provided in Fig. 4.

7 Stegomalware in Windows Applications

The applications have been found one of the convenient mechanisms for stegomalware to intrude into the systems. We have demonstrated the hiding of malicious JPEG images within Windows applications to evaluate the detection rate of several commercially available anti-malware scanners toward the stegomalware in Windows applications. Specifically, we have selected Windows operating system since to the best of our knowledge, there doesn't exist any such work or demonstration for the Windows platform relative to Android [10–13]. Concerning the hiding of malware in

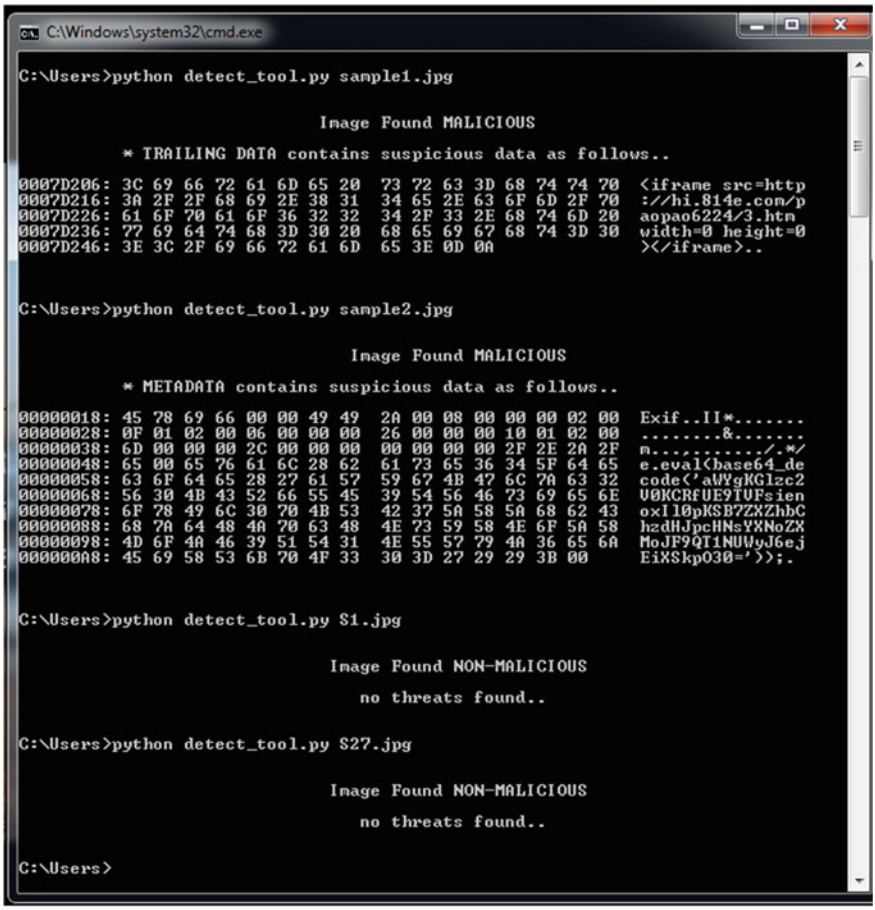

Fig. 4 Screenshot representing an instance of the output of the tool proposed

apps using steganography, this is the first work highlighting this issue in Windows. Moreover, Windows has a large user base which makes it a potential target for malware attacks than others. To demonstrate, ten most frequently used Windows applications have been used. On the other hand, ten images have been randomly selected from the dataset of malicious images used in this paper. To enable hiding, a tool called Resource Tuner [35] has been used that allows editing the resources in Windows applications. We have used the ten images randomly, one each for each application, embedding and replacing the image resources within the applications used. The apps after hiding the images were scanned via VirusTotal. We intend to assess whether the applications containing stegomalware are detected by the anti-malware scanners. Figure 5 presents the number of anti-malware engines used by VirusTotal which detected these Windows applications containing malicious images

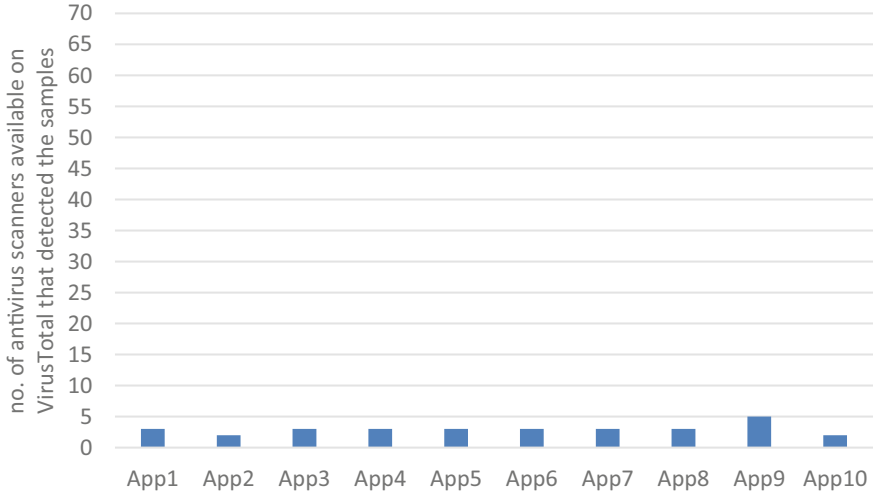

Fig. 5 Detection of Windows applications containing stegomalware

hidden. The figure shows that on average only 3 or 4 antiviruses out of over 70 scanners used by VirusTotal were able to detect such applications. This indicates the role of malicious image steganography in Windows applications to evade detection. This highlights a need for more effective Windows malware detectors since thwarting malware attacks on Windows is a formidable challenge.

8 Conclusion

This paper has proposed an effective tool to detect malicious JPEG images. Analyzing three types of JPEG images: malicious, benign, and the stego ones hiding non-malicious data, a low FNR and FPR with a better detection rate relative to state-of-the-art techniques have been attained for the 5,893 images. Indeed, the tool has classified the stego images as non-malicious, indicating the effectiveness of the tool in classifying the images hiding malicious and non-malicious data. Unlike existing works that have focused on finding steganography artifacts or used feature-based analysis which lacks revealing the hidden data, the proposed tool has attempted to locate malign data within images using certain keywords and produce that data as the output. This functionality has been found missing in the existing literature where hidden malicious data is not accessible for analysis. The proposed tool thus can be integrated with existing tools to enhance detection. Moreover, malicious images have been demonstratively hidden in Windows applications showing the role of steganography in apps to evade several antivirus scanners, indicating a need for more effective Windows malware detectors. Also, the scope of our algorithm can be extended in the future for other image formats, requiring the algorithm is modified/adapted accordingly to read format-specific data.

Acknowledgments All the authors have contributed to the work without any conflict of interest. The authors specifically thank VirusShare.com and Hybrid-Analysis.com for granting access to their malware collection. To mention, the research has not received any grant from any funding agency in the public, commercial or not-for-profit sectors.

References

1. Mazurczyk W, Caviglione L (2015) Information hiding as a challenge for malware detection. IEEE Secur Priv 13:89–93. https://doi.org/10.1109/MSP.2015.33
2. Beek C, Dinkar D, Gund Y, Lancioni G, Minihane N, Moreno F et al (2017) McAfee labs threats report: June 2017 (2017). https://www.mcafee.com/enterprise/en-in/about/newsroom/research-reports.html
3. Cabaj K, Caviglione L, Mazurczyk W, Wendzel S, Woodward A, Zander S (2018) The new threats of information hiding: the road ahead. IT Prof 20:31–39. https://doi.org/10.1109/MITP.2018.032501746
4. Mazurczyk W, Wendzel S (2017) Information hiding: challenges for forensic experts. Commun ACM 61:86–94. https://doi.org/10.1145/3158416
5. Aljamea MM, Iliopoulos CS, Samiruzzaman M (2016) Detection of URL in image steganography. In: ICC '16: proceedings of the international conference on internet of things and cloud computing. ACM, pp 1–6. https://doi.org/10.1145/2896387.2896408
6. Puchalski D, Caviglione L, Kozik R, Marzecki A, Krawczyk S, Choraś M (2020) Stegomalware detection through structural analysis of media files. In: ARES '20: proceedings of the 15th international conference on availability, reliability and security. ACM, pp 1–6. https://doi.org/10.1145/3407023.3409187
7. Amsalem Y, Puzanov A, Bedinerman A, Kutcher M, Hadar O (2015) DCT-based cyber defense techniques. In: Proceedings of the SPIE 9599, applications of digital image processing XXXVIII. SPIE. https://doi.org/10.1117/12.2187498
8. Carrega A, Caviglione L, Repetto M, Zuppelli M (2020) Programmable data gathering for detecting stegomalware. In: 2020 6th IEEE conference on network softwarization (NetSoft). IEEE, pp 422–429. https://doi.org/10.1109/NetSoft48620.2020.9165537
9. Pevny T, Kopp M, Křoustek J, Ker AD (2016) Malicons: detecting payload in favicons. In: Electronic imaging symposium, media watermarking, security, and forensics, pp 1–9. https://doi.org/10.2352/ISSN.2470-1173.2016.8.MWSF-079
10. Suarez-Tangil G, Tapiador JE, Peris-Lopez P (2014) Stegomalware: playing hide and seek with malicious components in smartphone apps. In: International conference on information security and cryptology, LNCS. Springer, Cham, pp 496–515. https://doi.org/10.1007/978-3-319-16745-9_27
11. Badhani S, Muttoo SK (2018) Evading android anti-malware by hiding malicious application inside images. Int J Syst Assur Eng Manag 9:482–493. https://doi.org/10.1007/s13198-017-0692-7
12. Cao C, Zhang Y, Liu Q, Wang K (2015) Function escalation attack. In: International conference on security and privacy in communication networks, LNICST. Springer, Cham, pp 481–497. https://doi.org/10.1007/978-3-319-23829-6_33
13. Suarez-Tangil G, Tapiador JE, Lombardi F, Pietro RD (2016) Alterdroid: differential fault analysis of obfuscated smartphone Malware. IEEE Trans Mob Comput 15:789–802. https://doi.org/10.1109/TMC.2015.2444847
14. Cohen A, Nissim N, Elovici Y (2020) MalJPEG: machine learning based solution for the detection of malicious JPEG images. IEEE Access 8:19997–20011. https://doi.org/10.1109/ACCESS.2020.2969022

15. Pérez JDJS, Rosales MS, Cruz-Cortés N (2016) Universal steganography detector based on an artificial immune system for JPEG images. In: 2016 IEEE Trustcom/BigDataSE/ISPA. IEEE, pp 1896–1903. https://doi.org/10.1109/TrustCom.2016.0290
16. Natarajan V, Sheen S, Anitha R (2012) Detection of StegoBot: a covert social network botnet. In: SecurIT '12: proceedings of the first international conference on security of internet of things. ACM, pp 36–41. https://doi.org/10.1145/2490428.2490433
17. Kunwar RS, Sharma P (2017) Framework to detect malicious codes embedded with JPEG images over social networking sites. In: 2017 international conference on innovations in information, embedded and communication systems (ICIIECS). IEEE, pp 1–4. https://doi.org/10.1109/ICIIECS.2017.8276144
18. Virus scanning website. https://www.virustotal.com
19. Fridrich J (2004) Feature-based steganalysis for JPEG images and its implications for future design of steganographic schemes. In: International workshop on information hiding, LNCS. Springer, Berlin, Heidelberg, pp 67–81. https://doi.org/10.1007/978-3-540-30114-1_6
20. Lin C-Y, Chang S-F (2001) A robust image authentication method distinguishing JPEG compression from malicious manipulation. IEEE Trans Circuits Syst Video Technol 11:153–168. https://doi.org/10.1109/76.905982
21. Cid DB (2013) Malware hidden inside JPG EXIF headers. Sucuri Blog, Website Security News. https://blog.sucuri.net/2013/07/malware-hidden-inside-jpg-exif-headers.html
22. Shah S (2015) Stegosploit—exploit delivery with steganography and polyglots. In: Briefings, Black Hat Conference. https://www.blackhat.com/eu-15/briefings.html
23. Khandelwal S (2016) Beware! Malicious JPG images on facebook messenger spreading locky ransomware. In: The hacker news, cybersecurity news and analysis. https://thehackernews.com/2016/11/facebook-locky-ransomware.html
24. Abrams L (2017) SyncCrypt ransomware hides inside JPG files, appends .KK Extension. Bleeping computer, Technology News Website. https://www.bleepingcomputer.com/news/security/synccrypt-ransomware-hides-inside-jpg-files-appends-kk-extension
25. Zahravi A (2018) Malicious memes that communicate with Malware. Trend Micro, IT security company. https://www.trendmicro.com/en_us/research/18/l/cybercriminals-use-malicious-memes-that-communicate-with-malware.html
26. Osborne C (2019) LokiBot malware now hides its source code in image files. ZDNet, Technology News Website. https://www.zdnet.com/article/lokibot-information-stealer-now-hides-malware-in-image-files
27. Szappanos G, Brandt A (2019) MyKings botnet spreads headaches, cryptominers, and Forshare malware. Sophos, cybersecurity company. https://news.sophos.com/en-us/2019/12/18/mykings-botnet-spreads-headaches-cryptominers-and-forshare-malware
28. Malware repository. https://virusshare.com
29. Malware analysis service. https://www.hybrid-analysis.com
30. Image database. http://sipi.usc.edu/database
31. Photos and Videos sharing platform. https://www.pexels.com
32. Social networking site. https://www.facebook.com
33. Verma V, Muttoo SK, Singh VB (2019) Enhanced payload and trade-off for image steganography via a novel pixel digits alteration. Multimed Tools Appl 79:7471–7490. https://doi.org/10.1007/s11042-019-08283-9
34. Westfeld A (2001) F5—a steganographic algorithm. In: International workshop on information hiding, LNCS. Springer, Berlin, Heidelberg, pp 289–302. https://doi.org/10.1007/3-540-45496-9_21
35. Visual resource editor. http://www.restuner.com
36. Butani B, Kumar Shukla P, Silakari S (2014) An exhaustive survey on physical node capture attack in WSN. Int J Comput Appl 95:32–39. https://doi.org/10.5120/16577-6265

37. Bhatt R, Maheshwary P, Shukla P, Shukla P, Shrivastava M, Changlani S (2020) Implementation of fruit fly optimization algorithm (FFOA) to escalate the attacking efficiency of node capture attack in wireless sensor networks (WSN). Comput Commun 149:134–145. https://doi.org/10.1016/j.comcom.2019.09.007
38. Kumar SA, Sinha S, Shukla P (2018) Design and development of image security technique by using cryptography and steganography: a combine approach. Int J Image, Graph Signal Process 10:13–21. https://doi.org/10.5815/ijigsp.2018.04.02