International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Design of Application to Detect Images Embedded with Malicious Programs

Robert T. R. Shoniwa[1], Geogen George[2]

[1]Information Security and Cyber Forensics, SRM University, India

[2]Information Technology, SRM University,India

Abstract: ***In today's world, malware can be propagated to victim systems in an increasingly diverse number of ways. One of these methods involves the passive distribution of malware by embedding in JPEG images which goes on to highlight that even simple images can be manipulated maliciously by criminals. The aim of this paper is to design an application that partially acts as a steganalysis tool to scan, detect and notify the user of the presence of a payload in either one or a set of selected images.it will then proceed to analyze the payload and verify whether it is a malicious program or not. It will also give a brief summarized file analysis of the detected payload. Ultimately, this will help highlight the need to consider images as a potential attack vector and then also offer a corresponding solution to this problem.***

Keywords: Steganography, Steganalysis, malware analysis, image analysis, image compression

1. Introduction

In July 2013 researchers at Sucuri [1] reported on an incident where they found an odd backdoor on a site that had been compromised. The oddity arose from the point that the said backdoor did not rely on the normal patterns such as base64 and gzipencoding to hide the contents contained within it. It actually stored its data within the EXIF header location of a JPEG image. In addition to that, it also used the two PHP functions to read the headers and then ultimately execute itself. This clearly illustrate that images can now be used as methods to try and compromise protected systems. However, an image on its own is relatively harmless but the moment a trigger is initiated; the image will immediately become an active participant in the malicious activity.

This Sucuri example highlights how malicious data can be ingeniously stored in the EXIF header of an image. This can lead to even more ways that JPEG images can be used maliciously. Specifically, this can be done by focusing on the point that, most antiviruses and Intrusion Detection Systems (IDSs) do not possess the facility to exercise both Steganalysis and file signature comparison with a virus signature database on images. It should be noted that the currently existing potential solutions to detecting malicious programs include the above mentioned IDSs and antiviruses. The benefit of the antivirus lies in that it has a virus signature database to scan files in a static manner and can also implement sandboxing to test suspicious programs and the way they operate.

However, the shortcomings of this methodology are clearer when it is dealing with a stego-image containing a malicious program. Unless the trigger has been activated, the image will remain an "innocent" image and not draw any suspicion. The benefit of IDSs lies in that they have the capability to scan incoming and outgoing packets at a host but the drawback is similar to that of antiviruses. The problem lies in that they both end up being mainly reactive and never proactive. This is because if they do create a file signature for an image, the same malicious payload may be embedded within another image and this will have its own new signature. All an attacker would need to do would be to continually switch the same malicious program among a large set of images. Hence, this brings about the need for the scanning tool being proposed in this project. It will be able to analyze an image file and retrieve any data it may be hiding and then ultimately hand it over to the antivirus for proper scanning if they are integrated together properly. This would overcome the problem where sandboxing and static analysis through signature based scanning do not detect anything malicious about the image.

The design will be described considering the following scenario as a premise. In this case, an attacker would have embedded a malicious program in an image using steganography, transferred it to a target host and then executed the malicious program. This leads to the implementation of the scanning application which will be used to counter the effects of the exploit scenario that has just been described and ultimately highlight the need for its use in future. The second section will comprise of a literature review, while the third section of the paper will proceed to explain the design of the proposed scanning application as well as a description of the proof-of-concept involving the actual embedding of a malicious program in an image. The fourth section will focus on the discussion of the propose design while the fifth section will offer the conclusion on the topic and future changes that can be made to the proposed application design.

2. Literature Review

In a paper published by Sajedi and Jamzad [2], they focused mainly on steganography methods. The paper discussed how due to the variety of contents found within images, the stego-images output by a steganography method are capable of possessing different and varying levels of detectability when they are scanned by steganalysis tools. This basically meant that a steganography method could result in statistical artifacts that are less detectable on some images compared to other images. By statistical artifacts they were referring to

any signs that are left or that are present on the image that can help prove or act as a sign that steganography has taken place. In addition to that, they analyzed different features of images to find the similarity between proper cover images for each steganography method they tested. Among those methods they listed were F5, Model-based steganography, Perturbed Quantization (PQ) as well as YASS which all manipulate some Discrete Cosine Transform (DCT) coefficients of images in order to embed secret data. They went on to discuss more about the ideal kinds of images to use that will leave the least traces of statistical artifacts after steganography. It aided in the selection of a steganography algorithm to implement in this project. The goal of this search was not to find an algorithm that was unbroken, because that would have made the Proof-of-concept even more difficult to implement and also lead us out of the scope of this project.

Methods to combat this were researched by Chamorro and Miyatake [3] who focused mainly on steganalysis of images embedded with data. Their paper stated that steganalysis is a technique that tries to detect some statistical evidence of hidden data in an image under analysis. Many of the steganalysers can detect stego-images generated by LSB steganography with a high detection rate. However if the stego-image is generated using JPEG steganography, these methods will show inefficiency to detect the presence of hidden message. It also stated that to select an efficient steganalysis method, some aspects must be considered. For example, false negative and false positive error rates are sufficiently small and independent of the amount of the secret message. In addition to that, the amount of features extracted from images must be as compact as possible. Some steganalysis methods were also highlighted include Difference Image Histogram Method (DH), Closest Color Pair Method (CC) and Wavelet Statistical Moments based Method (FE).

Yan and Ansari [4] wrote a paper which highlighted the aspect of unpacking obfuscated programs. It described how unpacking is the process of stripping the packer layer (or layers) of packed executables to restore the original contents so that antivirus programs and security researchers can inspect and analyze the original executable signatures. There are three different techniques to unpack a packed file which are manual unpacking, static unpacking and generic unpacking. The paper therefore highlights that antiviruses typically use static unpacking while others may use emulation to implement generic unpacking. Also, of the known unpacking methods, the more automated one that can be used by the scanner is static unpacking unlike generic that may put the system at risk and manual that needs frequent user interaction. Therefore static would be ideal for implementation within an application such as the one being proposed in the project.

3. Design

The system being proposed is a scanning application that can detect the presence of malicious programs in JPEG images, the extraction and reporting of the found data. In order to do that, as shown in Fig 1, a proof-of-concept will have to be done in order to prove that the threat is real and that images may be used as a potential attack method. The scanning application will primarily focus on the steganalysis of images.

The goal is to find out if any data is hidden in an image and then proceed to extract and analyze the data which could potentially be a malicious program. This method will only use a simple data-set in the proof of concept and resulting detection procedure. Standard and up to date enterprise virus signature databases will not be used but a facility that can possibly be used to enable future integration can also be noted. This design is comprised of three modules which are:

- Embedding module
- Execution module
- Scanning application

A. Embedding Module

The embedding module will firstly engage in the creation of the malicious program. First of all, the embedding module will involve the use of Msfpayload to generate a reverse-TCP payload and return the generated shellcode in Ruby language. It will also include setting the port on the attacking host that will be listening for the connection back from the target as well as setting its own IP address so it can be referenced by the target host. After that, the shellcode is then encoded using Msfencode. The malicious payload is then returned as an executable file after encoding and control is transferred to the obfuscation program, to help evade antivirus software. Some compression packers' unpacking process involves four consecutive steps which are modified LZMA (Lempel–Ziv–Markov chain algorithm) decompression, E8/E9 decompression, rebuilding of the import table and then ultimately jumping to the Original Entry Point (OEP) of the program. The compression and decompression as well as the fact that all this occurs in memory is how packers typically evade antiviruses.

This then assigns a new file signature to the malicious program and then transfers control to the steganography tool. Embedding of malware in stego-image can involve the use of the F5 algorithm [5]. F5 is a steganography algorithm for hiding information in JPEG images through manipulation of the Transform Domain steganography method which involves the use of Discrete Cosine Transforms (DCT). This is all through the use of the JPEG Lossy compression mechanism [2]. After the DCT is done and the quantization stage takes place, the embedding process occurs.

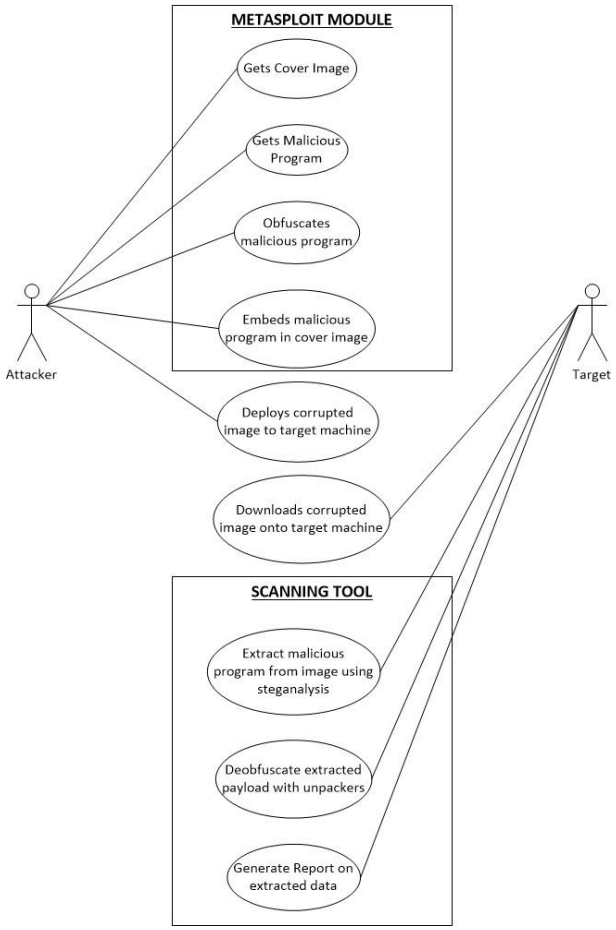

Figure 1: Use case Diagram of system actors and involved components

B. Extraction and Execution Module

The procedure for how the program will execute [4] is based primarily on how antiviruses operate. The obfuscation program changes the Original Entry Point (OEP) and therefore will return a different offset when the antivirus tries to locate offset A as normally expected. Due to this, the antivirus will not detect the malicious program's signature and will allow it to execute. In addition to that, the antivirus typically does not contain steganalysis or extraction tools for the analysis of steganography in images and therefore will not detect anything suspicious in the stego-image we would have created containing the malicious program. Therefore, when triggered, the stego-image will have the malicious program extracted from it and executed in memory. This program will then request a connection to the attacking host and then offer it command shell access to the target. The attack host will be running a multi-handler exploit from within the Metasploit framework which, in turn, will also be running a listener for the specified payload and port.

C. Scanning application

The scanning application will load the image to be scanned into memory. The image will then have a unique ID assigned to it. The goal of doing this is so that if the same image or a similar image with a different file name is scanned during this session, the scanner will not have to undertake the whole procedure again but just treat it in the same manner it did the preceding image with the same ID. This will help make the procedure quicker and more efficient. This is illustrated in the activity diagram in Fig 2. The steganalysis algorithms [3] (chi-square attack, visual detection, histogram analysis) will then be used to check if the image has any steganography artifacts. This refers to any signs or properties of the image that could be signs of the fact that steganography has been implemented on it. If this is not so, an Image Threat Level of 0 will be set due to the fact that there will be no sign of any steganography occurring on the image. The scanner will then attempt to extract the data from the stego-image and retrieve it. After that, it will analyze the headers of the retrieved data to check for magic numbers. Magic numbers are unique identifiers located in a file's header that describe the file type of the file itself.

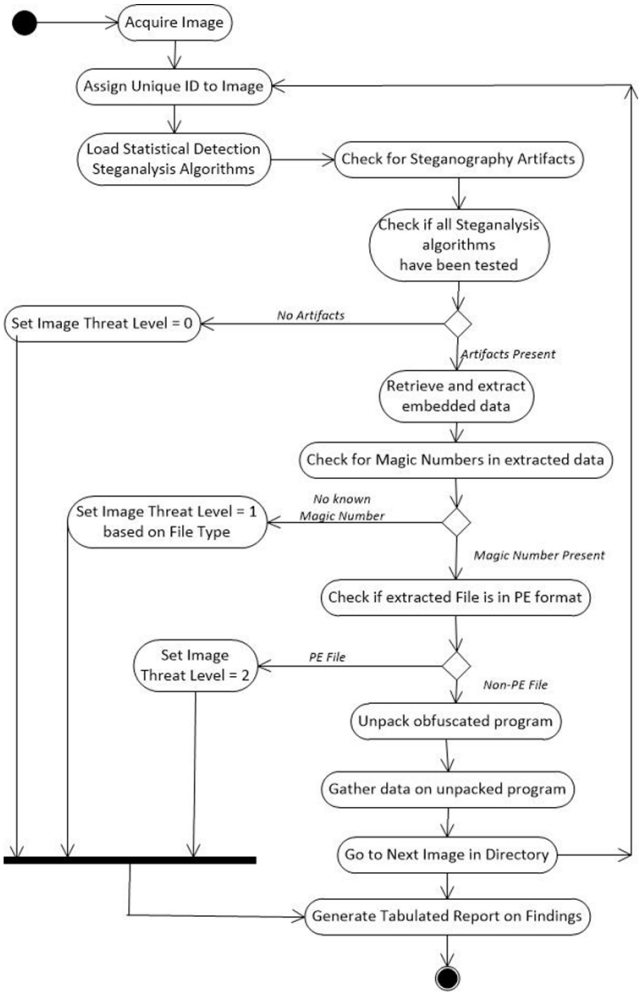

Figure 2: Activity Diagram for Scanning Tool

Typically, executable files such as .exe and .dll files will be the most suspicious. In that case, we will then set a threat level for images where non-executable files will be assigned a threat level of 1. This would basically mean that they are potential threats to the system but of an intermediary level due to the fact that they are not executable. If the data is in Portable Executable (PE) format then the image threat level is set to 2, the highest level. A command is then run on the extracted file and it will return the kind of packer used on the file as well as its basic properties such as size. A hash (or in some cases the extracted file itself) will also be submitted to Virustotal to know if the file is malicious or not. Ultimately, a report will then be generated with this data showing what the scanner found out after its activities.

4. Discussion

The proof of concept can be successfully simulated as having been deployed to the target machine via a compromised USB drive and then having the malicious program extracted and executed through the exploitation of the autorun.inf file on unpatched systems [13]. A hidden batch program will be initiated and used to extract the malicious program from the stego-image and then create a directory within the target system where it will then store the extracted malicious program. This will highlight the need for the proposed application. The goal of this project is to develop an application which is able to detect the presence of malicious programs within images as illustrated in the stages of the activity diagram in Fig 2. The design of the tool integrates a number of already existing steganalysis methods into the tool itself which included the statistical attack techniques Chi-square attack [6] and Histogram analysis attack [6]. The source code for these techniques can be gathered from already existing open source tools such as stegbreak/stegdetect [5] as well as porting of readily available Matlab modules' code to the tool.

When the tool is run on a set of JPEG images embedded with small malicious programs it will most likely report the presence of statistical artifacts. This can then be used as evidence of the high probability that the reported images were images containing hidden data. The application's test cases will include images that have used F5, Yet another Steganography Scheme (YASS), Outguess and JSteg. Of the four tools used, the only tool that will most likely manage to successfully bypass the first analysis stage of the tool is YASS. The reason is based on the fact that the first stage involves the use of Chi-square attacks and Histogram analysis in order to act as statistical methods to detect steganography in the images. The reason why YASS may not be detected is because it is known to be undetectable by most blind steganalysis attacks [6] [8]. This stage's efficiency can be further increased by adopting the method implemented by the Gargoyle [12] proprietary system which maintains a signature database of all steganography tools which can also help quicken the procedure of detection and extraction of data from stego-images by transforming the attack from a blind-steganalysis attack to a targeted attack towards a specific method.

After running all the statistical attack methods on the set of images, the application will then assign a threat level to each of the images and also generated a hash for them in the event that an image was encountered more than once to eliminate redundant processing. This will resulted in fewer images to process on in the second stage and increase the application's efficiency. This stage involves the use of feature extractors [8] which can also be ported from already available Matlab modules online [10]. The simplest stego-images to extract data from are typically JSteg due to the fact that it uses no key and anyone can extract the data [9]. Another optional method (which would be more costly) could have involved a statistically-targeted attack on selected steganography algorithms. This would involve running a dictionary attack on the set of images in order to try and acquire the passphrase or key used to embed data in the stego-image. Due to this, the second stage will probably take more time than other stages. However, the chief benefit lies in that if a set of images comes from one location (USB key drops or passive propagation through a folder on an ftp site); it is highly likely that the attacker will use the same or a similar key for the extraction of the hidden data.

After the extraction of data, the magic number for each of the extracted data will be analyzed as well. The application is specifically meant to target PE format files such as .dll and .exe files. However, the report generated will account for all extracted data as well so a set of all known magic numbers, will be used to help categorize the extracted data. PE files, even when obfuscated with tools such as Obsidium, Themida or any other packer, will still have the PE header containing the PE magic number which is 4D5A in hexadecimal. Therefore, the rest of the extracted files will be set to a threat level of 1 while those of PE format were set to level 2. Attempts will be made to unpack the programs if obfuscated but this will also be a processor-intensive procedure. In such a case, the use of the Taggant system [11] which maintains a database of all packers and their unique signatures could also help reduce the amount of time taken during this process. In addition to that, the extracted executables will also submitted for scanning to Virustotal to check if they contain known signatures associated with malicious programs. This can be further improved upon by submitting them all for dynamic analysis through sandboxing [4] in case their signatures are not known. The reason behind this is that due to the fact that they were hidden in the first place, they should be treated as potentially malicious.

Ultimately a report will be generated listing the findings as well as information from the Virustotal response. It should be noted that if the tool is to use targeted statistical steganalysis attacks on more than one steganography method then it would increase its effectiveness and also cater to a larger variety of stego-images created using other transform domain steganography methods. In the end, this would also go to prove that the proposed application could be useful in averting threats that may come in the form of images embedded with malicious programs.

5. Conclusion

In a nutshell, it can be seen that computer systems of today are facing danger from file types that would normally not be expected to carry malicious programs. In order to justify the need of the proposed application, a proof -of-concept had to be designed as well so as to emphasize the risks computer systems are facing. Ultimately, the proposed scanning application could help thwart most attacks arising from JPEG images embedded with images. In future, the application may also be integrated with antivirus software to make them even more efficient by passing the extracted programs to the antiviruses for sandboxing. It can also be integrated with the Taggant system to increase efficiency in the detection of the packers/obfuscation tools used. In addition to that, it could even be integrated with the Gargoyle system to help identify the steganography tool used to hide images thereby improving the overall speed and efficiency of the tool as well.

References

[1] D. Cid. (2013, July 16). Malware Hidden Inside JPG EXIF Headers [Online]. Available: http://blog.sucuri.net/2013/07/malware-hidden-inside-jpg-exif-headers.html, Accessed: 2014, August 29
[2] Sajedi, H., & Jamzad, M. (2010, March). Selecting a reliable steganography method. In Multimedia Computing and Information Technology (MCIT), 2010 International Conference on (pp. 69-72). IEEE
[3] Chamorro, A.G.H.; Miyatake, M.N., "A New Methodology of Image Steganalysis Including for JPEG Steganography," Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010, pp. 434, 438, Sept. 28 2010-Oct. 1 2010.
[4] Yan, W., & Ansari, N. (2009, August). Why anti-virus products slow down your machine?. In Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th International Conference on (pp. 1-6). IEEE.
[5] Fridrich, J., Goljan, M., & Hogea, D. (2003, January). Steganalysis of JPEG images: Breaking the F5 algorithm. In Information Hiding (pp. 310-323). Springer Berlin Heidelberg..
[6] Westfeld, A. and Pfitzmann, A. (2000) 'Attacks on Steganographic Systems',3rdInternational Workshop. Lecture Notes in Computer Science, Vol.1768. Springer-Verlag, Berlin Heidelberg New York
[7] Solanki, K., Sarkar, A. and Manjunath, B.S. (2007) 'YASS: Yet Another Steganographic Scheme that Resists Blind Steganalysis', 9th International Workshop on Information Hiding, Saint Malo, Brittany, France
[8] Johnson, N.F. and Jajodia, S. (1998) 'Steganalysis of Images Created Using Current Steganography Software', Workshop On Information Hiding Proceedings, Portland, Oregon, USA.
[9] Provos, N. and Honeyman, P. (2003) 'Hide and Seek: An Introduction to Steganography',Proc. IEEE.
[10] Fridrich J., Holub V., Denemark T., (2014, July). Feature Extractors for Steganalysis[Online],Availableat:http://dde.binghamton.edu/download/feature_extractors/, Accessed: 2014 September 12
[11] Lakhotia, A., & Phoha, V. V. (2012). (DEPSCOR FY 09) Obfuscation and Deobfuscation of Intent of Computer Programs. LOUISIANA UNIV LAFAYETTE.
[12] Kessler, G. C. (2004). An overview of steganography for the computer forensics examiner. Forensic Science Communications, 6(3), 1-27.
[13] Gonsalves A. (2012, November 30). Security Firms warn of spreading Windows Autorun malware [Online]. Available: http://www.csoonline.com/article/2132598/malware-cybercrime/security-firms-warn-of-spreading-windows-autorun-malware.html Accessed: 2014, September 3

Author Profile

Robert T.R. Shoniwa is student studying towards an MTech in Information Security and Cyber Forensics at SRM University, India. He also holds a BTech (with Honors) degree in Computer Science from Harare Institute of Technology in Zimbabwe.

Geogen George is a researcher in the Cyber Security Research Centre at SRM University. He also holds an MTech degree in Information Security and Cyber Forensics from SRM University.