Indicators of Malicious Activities in Exe's and PDF's

Joseph K A , Merrin J, Neethu Lakshmi G
ER & DCI-IT
Centre for Development of Advanced Computing (CDAC)
Trivandrum, India

Dija S
Resource Center for Cyber Forensics (RCCF)
Centre for Development of Advanced Computing (CDAC)
Trivandrum, India

***Abstract*—Malware analysis is a diverse field where it is becoming progressively difficult to keep continued track of malicious activities that deviate in their character and method of operation. In this paper we point out strong indicators that will help us to flag an executable and PDF file as being malicious or not. We make use of the Portable Executable (PE) file format and PDF format to explore the insides of executable and PDF's respectively. Closely observing the files have given us an insight into data structures and their attributes that help us with our purpose. We have also included substantial pointers that will help in the implication of malware writers in the court of law. These observations are utilitarian to a forensic investigator who has to deal with a legion of files on an individual system by constricting them down to a few files with striking probabilities of malicious activity.**

Keywords — Malware; Artifacts; Executables; PDFs; Forensics

I. INTRODUCTION

In the interconnected world of computers, malware has become an omnipresent and dangerous threat. Given the devastating effect malware has on our cyber infrastructure, identifying malicious programs is an important goal. With technology growing at its pace, criminals are making extensive use of malwares to control computers and to steal personal and confidential information for profit. To combat these malwares a branch of cyber forensics called malware forensics was developed. Malware forensics deals with analyzing a malicious code may be a script or an executable, to identify the illegal activities and purposes.

Malware analysis can be done using static analysis and dynamic analysis. By using different malware analysis techniques it helps the investigator to identify the correct intention of a particular sample [8]. Static analysis is a code analysis method and dynamic analysis is a behavioral analysis method. Code or static analysis is viewing the code and going through it to get a better understanding of malware and what it does. Behavioral analysis is how it behaves when executed, what interactions take place, what gets installed and if any code is executed without permission. When performing malware analysis an investigator should perform both static and dynamic analysis to get a better understanding of what a malware does. Mostly the malware resides as an executable as well as scripts in documents specifically in the case of portable document format (PDF) files.

II. STRUCTURE OF FILES

Every executable starts as a set of source code written in any particular programming language and goes through a few organized steps before it starts its execution in the physical memory as shown in Fig. 1. The programming language can be of the authors choice. The resultant source code is then compiled to give a binary output in the form of object code which exists as object files with .obj extension. The linker then takes the output of the compiler and the required dll's and stitches them together to give us the executable file. It is the responsibility of the loader then to place the executable file in the physical memory for its execution. The execution can begin after the loading process is complete [8].

A. Portable Executable Files

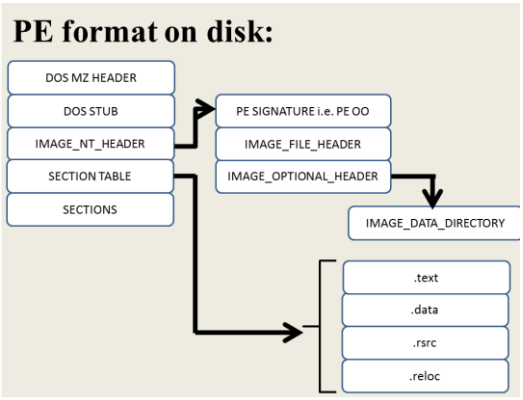

Fig. 1. Portable Executable Structure on disk

A robust understanding of the PE file format shown in Fig. 1 of a suspect executable program that has targeted a Windows system will best facilitate effective evaluation of the nature and purpose of the file. The basic supposition of this study is that there should be some distinctive characteristics between malware and benign programs since they are built in different intention. The Portable Executable (PE) format is a file format used in 32-bit and 64-bit versions of Microsoft Windows operating systems for executables, object code, and DLLs.

B. PDF Structure

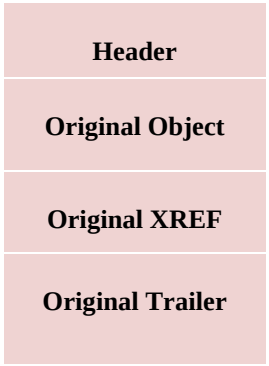

Fig. 2. Initial structure of a PDF file

PDF is the short for portable document format. PDF is one of the widely used applications for sharing and viewing documents. Solid understanding of the PDF file structure as shown in fig.2. is very helpful to analyse a malicious PDF file effectively. A PDF file is often a combination of vector graphics, text and raster graphics, which can represent themselves as rich text, drawings, either two or three dimensional images or multimedia such as audio, video or Flash. In such a way PDF has an incredible feature set.

III. FORENSICALLY RELEVANT ARTIFACTS

A. Artifacts in executables

1) IMAGE_DOS_HEADER

a) e_magic: In a PE file, the e_magic attribute of IMAGE_DOS_HEADER signifies a valid DOS header.e_magic contains the value 0x4D 5A.

b) e_lfanew: e_lfanew is a four byte attribute which is located at the end of the DOS header. It contains the offset of the PE header, relative to the file beginning.

2) DOS Stub

The DOS stub is a MS-DOS 2.0 compatible executable written in assembly language which always consists of a small number of bytes that output an error message. In a Win32 system the PE loader just skips the following DOS Stub. This section is not mandatory for the proper working of an executable. Hence, this location can be exploited by malware authors to embed malicious codes.

3) IMAGE_FILE_HEADER

a) No. of sections: Another useful entry in the IMAGE_NT_HEADER structure is the no. of sections field which is two bytes in size. It is important to know how many sections are there in a PE file, more specifically, how many section headers and section bodies. Each section header and section body is laid out sequentially in the file, so the number of sections is necessary to determine where the section headers and bodies end. In addition to the sections that exist by default in a PE file, it is possible to add other sections too. However, the Windows loader limits the number of sections to 96. This possibility can be exploited by the malware authors to include malicious code in a PE file.

4) IMAGE_OPTIONAL_HEADER

a) Size of code: This value lets us know how big the code section is. If there are multiple code sections then it gives the combined size of all these sections. If a malware author happens to add another code section so as to be able to execute his own malicious code then this size will not be in compliance with the expected value.

b) Address of entry point: This is the pointer to the location for the starting point of instructions that will be executed when the executable is loaded into memory. In case of packers this will point to the decryption code that will unfold the original compressed code. Malware writers tend to change the address of entry point so that they can redirect the flow of execution.

c) Size of image: It gives the size of the loaded executable in memory which includes the headers and the sections, so then if someone tries to modify the headers or add extra sections then it will show a different size.

d) Size of headers: This value includes the combined size of the headers and the section table so then if we have the total size, subtracting Size of headers from it will give a value that should be consistent with the size of contents after the section table. Any deviation indicates a possible modification of the executable.

e) Checksum: The checksum includes the dll's that will be loaded by the executable at run time and so if a call is made to a malicious dll then the checksum will change

5) IMAGE_IMPORT_DIRECTORY

Malware programs require system calls to be invoked to interact with the OS in order to perform malicious actions. Therefore, analyzing and extracting malicious behaviors from these programs require the identification of invoked system calls. Besides the predefined mechanism of system calls that require trapping to kernel, application programs may interact with the operating systems via higher level shared helper modules. Most malicious

programs use Win32 API which is a collection of services provided by DLLs that reside in user space. These details are revealed by the Import Directory.[5,6,7] The Import Directory is actually an array of IMAGE_IMPORT_DESCRIPTOR structures which contains a list of imported dlls used by the executable. Each structure is twenty bytes and contains information about a DLL which the PE file imports functions from. Number of IMAGE_IMPORT_DESCRIPTORs should always be equal to the number of dlls used in the executable. If there is any mismatch in their number it is an indication that the executable file under consideration is suspicious. IMAGE_IMPORT_DESCRIPTOR structure has an attribute Name1 which contains a pointer (RVA) to the ASCII name of the DLL. There are two methods by which a dll can be imported: import by name and import by ordinal. If we are using import by name then the important attribute is 'Name1'which is one byte in size and contains the name of the imported function. The name is a null-terminated ASCII string. If we are using import by 'ordinal' then the important attribute is ordinal which is of size four bytes and this attribute is located inside IMAGE_THUNK_DATA32 structure.

6) IMAGE_SECTION_HEADER

Since there is one such data structure each for every section present, the information here reveals a lot about each one

a) Name: Some packers and malcodes add their own sections. These names might ring a bell if they have been noticed previously. For example packers like UPX, ASPACK create sections .upx and .aspack respectively.

b) Size of raw data: It shows the size of the initialized data but if it is less than Virtual size then the rest of it is filled with zeros. This sequence of zeros can be replaced by malware authors to fit in their malicious code.

c) Pointer to raw data: This gives us the starting of the first page of the particular section. This could be edited to redirect the section to any other content. If it is an executable section and someone tries to lead it to a malicious script then it could lead to unwanted damage

d) Characteristics: This could decide whether a particular section is supposed to be executable or not and is supposed to contain only data items either initialized or uninitialized. A malware can overwrite a particular section or an empty space that initially is not executable but then can edit the Charecteristics flag to make it executable. This could go undetected easily.

7) PE File Sections

As mentioned every section has its own characteristics. Any deviation from the normal standards could be indication of something suspicious. Below are a few standard sections but any section that comes up in an executable should be checked for its attributes.

a) .bss: The .bss section represents uninitialized data for the application, including all variables declared as static within a function or source module

b) .rdata: The .rdata section represents read-only data, such as literal strings, constants, and debugs directory information

c) .rsrc: The .rsrc section contains resource information for a module.

B. Implicating a Malware Writer

A few attributes could help the case when a malware writers contribution to a particular crime is to be proved. These are not detrimental in nature and can be used as supportive facts to help the cause of the prosecution.

1) IMAGE_FILE_HEADER

a) TimeDateStamp: This can help us to know when the malware writer created the executable file. It is the time when the linker did its job. The time and date are represented as the number of seconds that have passed since the midnight on 1st January,1970.

2) IMAGE_OPTIONAL_HEADER:

a) Linker version: The linker used by the malware author in the creation of his executable. It could be checked with the one present on his system.

C. Artifacts in PDF

1) File Header

The file header consists of 1024 bytes and it should starts with "%PDF1.X". This header or magic number can be placed anywhere in this 1024 bytes[2]. So this space can be used to include some other hexadecimal data or strings and can include some other header (malformed header) to make investigation tricky and to make the investigator consider it as some other document. In the analysis of a document we should always consider the first 1024 bytes instead of considering only the first 4 bytes.

2) File Body

All the objects inside a PDF should be checked. Indirect objects in the body should be specifically considered. Because indirect objects[3] are the objects that may refer other objects, so all the relations between the objects should be derived to get the actual logical structure of a PDF file. Basically labeled objects are called indirect objects. Another object considered in investigation is the stream objects. These are represented in between stream and endstream. The streams will be mostly encoded. The entries that are common to all streams are length, filter, decodeParms, F, FFilter, FDecodeParms, DL. The stream objects may be compressed or mostly encrypted and can

hide data using a single filter or combinations of filters. The filters are widely used to hide the javascripts. There are many actions that must be considered. Many obfuscations techniques can be used to hide the data. Some of the obfuscation techniques include simple obfuscation, split strings, using regular expressions etc.

3) Cross-Reference Table

The cross reference table is the important part of the PDF that is analyzed by the investigators and also this is the tedious task for the investigator since all the objects have to be identified. The problems that arise in the case of cross-reference table that makes it suspicious is that sometimes the number of objects specified in the subsection of cross-reference table will not match the objects in the file. Somtime if *the line feed* character is missing then it may result to 19 byte long instead of 20 bytes.

4) Generation Number

All the indirect objects have a unique identifier**.** This object identifier consists of two parts. A object number and a generation number[3]. Object number are numbered sequentially in normal case but this is not required it can be assigned in arbitrary order too. In a new file the generation number is normally 0, non-zero generation numbers are created when that object is updated. Objects with generation numbers other than zero must be considered. The maximum generation number is 65535.when a cross reference table reaches this maximum value that object is never used.

5) Incremental Updates

The PDF file can be updated without rewriting the file. Any changes applied to the file are appended at the end of the file. The main advantage of such incremental update is that the changes to a particular document are saved quickly. In versions before 1.4 was not able to use an incremental update. In an investigation this has to be clearly checked because in normal files more than one incremental updates are not done. So if we are conducting a forensic investigation or malware analysis don't stick the analysis only onto the final version of the document. If there is a presence of two or more incremental updates then that particular file should be verified to find anything unusual there. A PDF with incremental update[4] contains header, original object, original cross-reference table, original trailer, updated object, updated cross-reference table and an updated trailer.

6) Metadata

The metadata[3] of a particular file must be considered in investigation. Most of The time the attacker will not include the metadata inorder to reduce the size of the file [1] or can modify the metadata using plugins. The metadata information can be got from the trailer "info" dictionary. It will not contain any private content or structural information. The metadata of the file can help find out the following things of a PDF file:

a) Title: The documents title.

b) Author: the name of person who created the document.

c) Subject: the subject of the file.

d) Keywords: keywords related to that document.

e) Creator: the name of the application that created the document and also informs whether it was converted from any other format.

f) Creation date: the date that PDF was created. The creation date should be taken into consideration. The format of date is D:YYYYMMDDHHmmSSOHH'mm'. This is in human readable form. It the date format is not as specified above then it can be considered as something unusual in the document.

g) Modification date: the date that the PDF file gets modified.

All these metadata have to be considered when analyzing a PDF file, Most of The time the attacker will not include the metadata in order to reduce the size of the file or can modify the metadata using plug-in**.**

7) Document Structure

A PDF document can be considered as a hierarchy of objects. Most of these objects are dictionaries and the root dictionary will be a document catalog dictionary.

a) Document catalog dictionary: The document catalog dictionary[3] is located from the root entry in the trailer of a particular PDF file. It contains information about how the document should present in the screen. The document catalog dictionary must be considered in the investigation point of view because it gives many necessary information needed like number of pages, version, type etc. The entries that will be available in the dictionary includes the following:

b) Type: the type of the PDF object that particular dictionary describing about.

c) Version: the version of the PDF specification.

d) Pages: gives the number of pages, parent, kids and the count. This can be modified by an attacker so that there will not be any connected paths from one page to another page and also they will not be well formed so that lot of missing paths will be seen.

8) Keywords

In addition to the structural elements of a PDF, there are embedded entities for investigative consideration, such as dictionaries, action type keywords.

a) Open action: the action to be performed when a document is opened.

b) AA: the additional actions to be performed when a document is opened.

c) URI: it contains the document level information about the uniform resource identifier.

d) Metadata: contains the metadata of the document other than from the info dictionary.

IV. FORENSIC RELEVANCE

Every application that we run on our computer whether it is a word processor, a spread sheet or a web browser begins with an executable file. Since executables form the basis of every application, there is a possibility of malware attacks on executables. The way an executable file is compiled and linked by an attacker often leaves significant clues about the nature of a suspect program. The basic supposition is that there should be some distinctive characteristics between malware and benign programs since they are built in different intention. Thus, it is a good reason to detect vicious binaries by the differences of natural specialties recorded in PE files. A thorough and deep understanding of the information available from the portable executable file format can help us to understand the malicious behavior of malwares in an executable.

Malware as we know have to reveal their true nature when in memory and this is because it is the only way they can run. Working on a memory dump of such a malware will be more enlightening than a simple static analysis of the concerned executable found on disk. Executable follow the same PE format in memory as found in disks with minor modifications and so we can make use of the PE format for the analysis of dumped files. This becomes increasingly important in sight of malware that exist only in memory and not on disk, thereby hardly leaving a trace in case of disk forensics. Static analysis can provide complementary insights to dynamic analysis in those occasions where obfuscations can be sufficiently overcome. Static analysis offers the potential for a better assessment and correlation of code and data of the program. Analyzing the system calls and APIs, control flow analysis, and tracking data segment references, it is possible to infer malicious system interactions and behavior of the executable under consideration. Static analysis, when performed on a deobfuscated executable can complement and even support dynamic analysis with a clear picture of the program logic.

Nowadays malware authors use a large amount of resources to create softwares that are dangerous and efficient. Targeted attacks make use of specifically crafted documents owed to their versatile functionality. So such a wide use of malware to commit and conceal crimes makes the digital investigators to make use of different types of malware tools and techniques. In order to fully understand the forensic relevance of the tool with above mentioned artifacts we can consider an example, consider a situation where we have a drug seller as a suspect. During the case investigation the investigator had to analyze the suspects e-mail. He found an e-mail with an attachment which is expected to have the details of drug suppliers. Since it is a e-mail attachment there is a chance of malware specimen. So, before going further into the evidence collection part by opening the attachment it is wise to analyze the file to detect the presence of suspicious content before it affects the workstation**.**

V. FORENSIC CHALLENGES

If the executable is packed using UPX or ASPack analysis becomes a bit complicated. Packing of malware is a major issue for the analyst since any static analysis of packed code is almost entirely useless. These packers will compress the executable and append the code with a stub containing the decompression algorithm. The entry point of the executable will be changed to the start of the stub and after the stub has done its job, execution jumps to the original entry point to execute the unpacked program. We can get the actual code that the malware author intends to run by taking a memory dump of that process because it will have to unfold itself in the main memory to be able to be executed by the processor. The executable will not run straight away because the dumped file will have its sections aligned to memory page boundaries rather than file alignment values. The only solution is to let the stub decompress the executable in memory and then dump the memory region to a file to get an unpacked copy of the exe and then perform analysis on it.

Another issue that arises when we try to conduct analysis of process memory dumps is that we might not find all the pages in the dump. The loader will load the pages that are only necessary for the execution of the code and the other pages might be loaded if required later. Even if all pages are loaded some of them might be moved to the page file to make some extra space in the memory. This leaves us with an incomplete executable that might not be exactly helpful if we intend to make use of the PE format for our analysis, since some of the data structures and the related content might be missing or spread across pages.

The analysis of PDF using the above mentioned artifacts is effective in malware analysis of documents. But there are some challenges in the examination of a PDF document. If a file is encrypted then we are not able to analyze that particular file unless the exact key value is known. Also if the file is password protected then it will be difficult for the investigator to analyze the file. Files with multi generation number are also a challenge still. PDF's with generation number 0 are analyzed by all the known PDF analysis tools.

VI. CONCLUSION

The artifacts that have been discussed here are useful signs about the activities of a file whether malicious or not. These indicators will help a malware analyst to sort out potentially harmful executable and PDF files from a large set of files as found on a hard disk. The importance lies in the fact that the scope of search for the investigator is reduced to a few number of files from a large set that might be initially provided. The challenges that have been encountered during both static analysis from disk and memory provide us an insight into how much information we can derive from a given file. Some attributes have been mentioned in case of executable which will help in supporting the case in court against an alleged malware author.

The prevalence of malware is increasing on the internet and is a threat to the today's world. There are many antivirus packages and tools that deal with different type of attacks. Document like PDF are often considered trusted and there are no effective tools which deal with attacks related to them. This paper discusses some forensically relevant artifacts that can be considered during the investigation to get the better understanding of the nature of malware and will prevent the workstation from such malware attacks.

ACKNOWLEDGMENT

We would like to express our gratitude towards the Resource Centre for Cyber Forensics, CDAC for giving us the opportunity to work on this engaging project and guiding us by providing the necessary support. We would also like to thank ER&DCI-IT for being instrumental in the work done by us.

REFERENCES

[1] Nedim ˇSrndi´c and Pavel Laskov : Detection of Malicious PDF Files Based on Hierarchical Document Structure, ACSAC '11 Dec. 5-9, 2011, Orlando, Florida USA Department of Cognitive Systems University of T¨ubingen T¨ubingen, Germany.

[2] Caglar Ulucenk, Vijay Varadharajan, Venkat Balakrishnan: Techniques for Analysing PDF Malware. 18th Asia-Pacific Software Engineering Conference Information & Networked Systems Security Research Faculty of Science, Macquarie University, Sydney, Australia, 2011.

[3] Adobe Systems Incorporated, 2006. PDF Reference, sixth edition – Adobe Portable Document Format Version 1.7.

[4] Didier Stevens, "Malicious PDF Documents Explained", IEEE Security & Privacy, Jan/Feb 2011, Vol 9(1), pp 80-82.

[5] Goppit : Portable Executable File Format – A Reverse Engineer View.Code Breakers Magazine (2006)

[6] . Microsoft Corporation: Microsoft Portable Executable and Common Object File Format Specification.Microsoft Corporation Revision 6.0 (1999)

[7] MattPietrek :An In-Depth Look into the Win32 Portable Executable File Format. MSDN Magazine, Microsoft Corporation. (2002)

[8] EoghanCasey, James M.AquilinaCameronH. Malin ,Malware Forensics: Investigating and Analyzing Malicious Code.Syngress(2008)

[9] Davide Maiorca, Giorgio Giacinto, and Igino Corona : A Pattern Recognition System for Malicious PDF Files Detection, P. Perner (Ed.): MLDM 2012, LNAI 7376, pp. 510–524, 2012. !c Springer-Verlag Berlin Heidelberg 2012Department of Electrical and Electronic Engineering (DIEE), University of Cagliari, Piazza d'Armi 09123, Cagliari.

[10] Elmar Gerhards-Padilla, Florian Schmitt, Jan Gassen: PDF SCRUTINIZER: Detecting JavaScript-based Attacks in PDF Documents, 2012 Tenth Annual International Conference on Privacy, Security and Trust, University of Bonn - Institute of Computer Science 4 Friedrich-Ebert-Allee 144, 53113 Bonn, Germany.