Analysis of Embedded Malware through Reverse Engineering

Vaibhav Gupta
Northern India Engineering College, F-26, Shastri Park
Guru Gobind Singh Indraprastha University
New Delhi, India
vaibhav12jan@gmail.com

Ashish Saxena/Rahat Sethi
AKS IT Services Pvt. Ltd., E-52, Sector-3, Noida
Utter Pradesh, India
ashish@aksitservices.co.in
rahat.sethi@aksitservices.co.in

***Abstract*— Antivirus Scanners are not able to detect about eighty percent [1] of malwares and thus cannot be relied upon. In this research paper we have presented a unified approach of analyzing embedded malwares in application data files like MS Word, MS Excel, Acrobat PDFs etc. After having analyzed about 103 malware, we found some typical characteristics used by malware to enhance their functionality and severity of attack and also avoid their detection from antivirus.**

Keywords- Malware analysis; Reverse engineering; Embedded malware analysis; Code analysis

I. INTRODUCTION

A malware is any executable content with unknown functionality specially designed to perform malicious activities without the owner's prior consent. Malware categories include: Viruses, Worms, Intrusion Tools, Rootkits and Spyware. Earlier malwares were only associated with executable files, but now these are found in popular application data files like MS-Word, MS-PowerPoint, MS-Excel, Acrobat PDFs, etc. These application data files are accessible to a major section of computer users rendering them vulnerable to malware infection.

A recent attack targeting the publicity of FIFA 2010 World Cup was in wild in which an infected PDF was circulated though spam emails. After opening the PDF, the malware tries to steal sensitive information, causes damage to the machine, monitor victim's activity, etc. We have analyzed about 103 different malware of different categories and functionalities and have concluded a unified approach for analyzing malwares through reverse engineering. In this research paper we will try and present a unified approach of analyzing malwares that are embedded in the application data files.

II. NEED FOR REVERSE ENGINEERING

Reverse Engineering is the art of analyzing a system, software or an object to its minutest detail to understand its functionality/operation principles. A Malware is reverse engineered to understand the working of a malware and the functionality and capability of the malware. The following are the main reasons to conduct reverse engineering of a malware:

-- Assess damage
-- Analyze malware functionality
-- Identify vulnerability
-- Catch the intruder
-- Prepare signatures

The main aim of reverse engineering malware apart from aforementioned is to understand the attacker's methodology and skill set. This helps in deeply analyzing the attacker and thus mitigating future attacks on the network along with planning the removal of malware from the network.

III. REVERSE ENGINEERING OF EMBEDDED MALWARE: A UNIFIED APPROACH

Embedded malware refers to the ability to hide malicious code inside a file and thus in most cases allowing it to pass through undetected by commercial antivirus. In this paper, we have presented a unified procedure that can be used to reverse engineer majority of the malwares, for required understanding & analysis of them which further aids in protecting/cleaning your network. Our approach will be broadly divided into the following major steps:

-- Extract the executable content
-- Check for Packers/Protectors/Cryptors
-- Unpack the executable content
-- Load into debugger & start the analysis

A. Environment for Carrying out Analysis

A malware when analyzed is required to be executed and reverse engineered. Upon execution the malware might alter system files or even make certain changes in the system. Hence it is extremely important to set up a sanitized environment for analyzing and reverse engineering the malware. A "Sanitized Lab" can be created to perform the analysis. The prerequisites of the lab are:

-- Two machines, one for execution of infected file & the other for analyzing any network traffic being sent upon execution. Both machines are required to be placed in an isolated network.

-- Two machines are required to be loaded with fresh operating systems, to make it easier to understand any new instance of running processes or changes made on the machine upon execution of malware.

-- Necessary tools for carrying out the examination.

B. *Virtualization of Environment for Carrying out Analysis*

In most scenarios, one can make use of Virtual Machines for carrying out analysis, thereby avoiding the requirement for an isolated infrastructure. But virtualization may be sometimes detected by malwares and they can change their behavior accordingly, hence an isolated infrastructure is advised.
Once the environment has been created the infected file is loaded on the system and analysis can now begin.

IV. Procedure for Carrying out Reverse Engineering of Embedded Malware

As mentioned earlier in the paper, the approach for carrying out reverse engineering would broadly be divided into four states. They are described as under:

A. *Extract the Executable Content*

The first step in analyzing an embedded document is to segregate the document from the portable file attached to it. Hence it is important to extract the executable content of the data file. In order to do so, the user should understand the files that are being created upon execution. This can be achieved with the aid of several tools that take system snapshots of file structure before and after the infection and will compare and produce the result containing the new files that were created. After you get the new files, you would be required to analyze each and every Portable Executable (herein referred as PE) file that is created upon execution of the original document/pdf. Some of the common extensions are: .exe, .dll, .drv, .sys, .ocx, .scr, .cpl.

B. *Check for Packers/Protectors/Cryptors*

About 79% of malwares are obfuscated with Packers/Protectors/Cryptors [2]. This technique is used by the malware to evade signature based detection techniques and also make it difficult for reverse engineers to reverse the file. Each packed executable is equipped with a loader stub which is responsible for restoring the packed executable into its original state at run time in memory. Most popular packers include UPX, Armadillo, aSPack, ASProtect. The tool we can use to check the packer being used is PEID [3]. When an executable is loaded in PEID, it displays the name of the packer that is used by the executable. Since unpacking process is different for every packer, PEID reveals useful information to initiate unpacking.

C. *Unpack the Executable Content*

A packed PE cannot be reverse engineered; hence the PE would be first requiring unpacking. In this paper we would be discussing a generic approach; detailed approach for unpacking would not be taken up in this paper. In general, the packer's loader stub unpacks the executable in memory and the Instruction Pointer (herein referred as IP), during the execution of the packed executable so that it points to the Original Entry Point (herein referred as OEP) to initiate the normal execution of the PE file.

The above inference is derived from the fact that sooner than later the packer is required to jump to OEP to transfer control to the normal executable. The OEP is required by the program to be called to transfer the execution and thus (generally) stored at run-time in Extended Stack Pointer (herein referred as ESP) register. We can load our executable in a debugger and trace the first change in the value of ESP register to find the OEP of the executable. After we have found the OEP, we can replace the EP with it in the 'PE headers' section. We can also use some automated tools for unpacking like: Quick Unpack, rl!depacker , GUnPacker.

D. *Unpack the Executable Content*

A debugger or debugging tool is a computer program that is used to test and debug other programs (the "target" program). Typically, debuggers offer more sophisticated functions such as running a program step by step (single-stepping or program animation), stopping (breaking) (pausing the program to examine the current state) at some event or specified instruction by means of a breakpoint, and tracking the values of some variables[4]. An unpacked (PE) executable (dll, exe, drv, sys, cpl, scr or ocx) is loaded into debugger for analysis.

After the PE is loaded into debugger, the executable is dissembled into basic assembly level instructions. Now we need to analyze the PE by executing it in the controlled environment of debugger. We will be placing breakpoint on each and every function call that will analyze for its reaction on the system. A breakpoint is a signal that tells the debugger to temporarily suspend execution of your program at a certain point. When execution is suspended at a breakpoint, your program is said to be in break mode.

Entering break mode does not stop or end the execution of your program; execution can be resumed at any time [5].
At first, we are required to analyze the Windows File handling Application Programming Interface (herein referred as API) calls to gather information about the files being created/read/changed. If the malware is creating some logs, like in case of keylogger, its activity can be traced easily. Some of the popular File handling APIs are:

- *File Handling API`s*

  - WriteFile()
  - CreateFile()
  - ReadFile()
  - FindFileName()
  - DeleteFile()

Malwares manipulate registry data to install themselves as startup object, hide their existence, disable some utilities like Task Manager, Registry Editor that may disable the malware, etc. We are required to check for all the registry manipulations that may have caused anomalies in the system. Some of the Registry Handling APIs are:

- *Registry Handling API`s*

  - RegCreateKey()
  - RegOpenKey()
  - RegDeleteKey()
  - RegDeleteValue()
  - RegQueryValue()
  - RegSetValue()

Majority of malwares communicate with their command and control center to receive controlling commands for the malware and send confidential information of the victimized machine at regular intervals. Adding breakpoints to Windows APIs responsible for network communication will help analyze the data being traversed through the network due to the malware infection. Some of the Network communication APIs are:

- *Network Communication APIs*

  - WSAStarup()
  - recv()
  - send()
  - InternetConnect()
  - InternetOpen()
  - InternetReadFile()
  - InternetOpenURL()

Some advanced malware coders incorporate some special anti-debugging tricks to avoid reverse engineering enumerate the functionality of their malwares. These techniques hinder in malware analysis, thus researchers are delayed in developing the antivirus signatures of such malwares thus increasing virus's life span

Some APIs, researchers need to check for anti-debugging tricks are:

- *Anti-Debugging APIs*

  - IsDebuggerPresent()
  - CheckRemoteDebuggerPresent()

Other APIs we that can also be checked for further analysis are as follows:

- *Miscellaneous APIs*

  - GetProcAddr()
  - GetAsynKeyState()
  - CreateProcess()
  - GetClipboardData()
  - GetStartupInfo()
  - GetSystemInfo()
  - WinExec()

V. Information Gathering & Analysis

Once the entire reversing of the malware has been conducted, it is required to gather and collate the information on each stage. This information is then collated and analyzed to ascertain the complete functionality of the malware. Several inferences can also be derived, such as files modified, connections made, traffic sent, registry entries made etc. Once all the information has been successfully gathered, documentation for the same should be made. With this info, malware should be removed from the affected machines successfully. Lastly signatures of the malware can be added into the security devices for identification & avoidance.

VI. Conclusion

This paper describes the generic approach that can be adopted while identifying embedded malware in various kinds of documents. The approach can be utilized for reversing any file and understand the functioning of the malware. This aids in sanitization of the network and also helps in predicting next generation of malware.

REFERENCES

[1]: http://www.zdnet.com.au/eighty-percent-of-new-malware-defeats-antivirus-139263949.htm

[2]Pedro Bustamante. Mal(ware)formation statistics http://research.pandasecurity.com/archive/Mal_2800_ware_2900_formation-statistics.aspx, 2007

[3]PIED, Generic packer Detector. http://www.peid.info/

[4] http://en.wikipedia.org/wiki/Debugger

[5]http://msdn.microsoft.com/en-us/library/4607yxb0.aspxG.

[6] "Malware analysis for fun and profit", http://www.windengineeringas.com/Malware_Analysis_for_Fun_and_Profit.pdf