Malware Analysis

Conference Paper · October 2014

DOI: 10.13140/2.1.4750.6889

Nirav Bhojani
Nirma University

Malware Analysis

Nirav Bhojani
Department of Computer Science and engineering
Institute of Technology, Nirma University
Ahmedabad, India
14MCEI05@nirmauni.ac.in

Abstract — ***Studies suggest that the impact of malware is getting worse. Two types of malware analysis are described here. One is Static Malware Analysis and other is Dynamic Malware Analysis. Static Malware Analysis has some limitations. So, Dynamic Malware Analysis is preferable for Malware Analysis. There are lots of tools available for Dynamic Malware Analysis. This paper includes all the stuff Limitations of Static Malware Analysis and tools of Dynamic Malware Analysis and Deobfuscating malware.***

Keywords — Malware, Virus, Worm, Trojan Keylogger, Malware Analysis, Static Malware Analysis, Dynamic Malware Analysis, Trends in Malware etc.

I. INTRODUCTION

Now a day, Internet becomes an essential part of the daily life of many people. On Internet many services are available and are also increasing day by day. More and more people are making use of these services. Online banking or advertising are the examples of the commercial services of the Internet. Just as in the physical world, there are people on the Internet with malevolent intents by taking advantage of legitimate users whenever money is involved. Malware like software of malicious intent helps these people accomplishing their goals.

II. WHAT IS MALWARE?

Malware stands for malicious software, designed to damage a computer system without the user's informed consent. Generally malware is categorized into following categories:

- *Virus:* It is a program that attaches itself to other programs in order to infect that program and perform some unwanted function
- *Trojan:* Trojan makes copies of themselves and steals information. It is standalone malicious program that does attempt to infect other computers in a completely automatic manner without help from outside forces like other programs.
- *Worms:* A worm is self replicated malware computer program which uses computer and network resources without authenticated user permission. In the network it is consuming the network bandwidth. This is security shortcoming on the target computer.
- *Spyware:* It is installed without a user's knowledge in order to report the behaviour of the user to the attacker.
- *Rootkit:* Rootkit is malware program which creates a backdoor into the system for the hacker's use, alters log files and destroyed the data files.

III. WHAT IS MALWARE ANALYSIS?

Malware analysis is the process of determining the purpose and characteristics of a given malware sample such as a virus, worm, or Trojan horse. This process is a necessary step to be able to develop effective detection techniques for malicious code. The tools used for malware analysis can basically be broken into two categories: static and dynamic (live). The static analysis tools attempt to analyze a binary without actually executing the binary. Live analysis tools will study the behaviour of a binary once it has been executed. Static and Dynamic analysis are described in detail in next sections. Automated malware analysis is a virtually intractable problem. It is simply not possible for one program to determine the exact behaviour of another program.

IV. *STATIC MALWARE ANALYSIS*

Analyzing software without executing it is called static analysis. Static analysis techniques can be applied on different representations of a program. Static analysis tools can also be used on the binary representation of a program. When compiling the source code of a program into a binary executable, some information gets lost. This loss of information further complicates the task of analyzing the code.

The process of inspecting a given binary without executing it is mostly conducted manually. For example, if the source code is available several interesting information, such as data structures and used functions can be extracted. This information gets lost once the source code has been compiled into a binary executable and thus impedes further analysis. There are different techniques used for static malware analysis. Some of are described below.

- *File fingerprinting:* Beside examining obvious external features of the binary this includes operations on the file level such as computation of a cryptographic hash (e.g., md5) of the binary in order to distinguish it from others and to verify that it has not been modified.
- *File format:* By leveraging metadata of a given file format additional, useful information can be gathered. This includes the magic number on UNIX

systems to determine the file type. For example from a Windows binary, which is typically in PE format (portable executable) a lot of information can be extracted, such as compilation time, imported and exported functions as well as strings, menus and icons.

- *AV scanning:* If the examined binary is well-known malware it is highly likely to be detected by one or more AV scanners. To use one or more AV scanner is time consuming but it becomes necessity sometimes.
- *Packer detection:* Nowadays malware is mostly distributed in an obfuscated form e.g., encrypted or compressed. This is achieved using a packer, whereas arbitrary algorithms can be used for modification. After packing the program looks much different from a static analysis perspective and its logic as well as other metadata is thus hard to recover. While there are certain unpackers, such as PEiD2, there is accordingly no generic unpacker, making this a major challenge of static malware analysis.
- *Disassembly:* The major part of static analysis is typically the disassembly of a given binary. This is conducted utilizing tools, which are capable of reversing the machine code to assembly language, such as IDA Pro. Based on the reconstructed assembly code an analyst can then inspect the program logic and thus examine its intention.

The main advantage of static malware analysis is that it allows a comprehensive analysis of a given binary. That is, it can cover all possible execution paths of a malware sample. Additionally, static analysis is generally safer than dynamic analysis as the source code is not actually executed. However, it can be extremely time-consuming and thus requires expertise.

V. *LIMITATIONS OF STATIC MALWARE ANALYSIS*

Generally, the source code of malware samples is not readily available. That reduces the applicable static analysis techniques for malware analysis to those that retrieve the information from the binary representation of the malware. Consider, for example, that most malware attacks hosts executing instructions in the IA32 instruction set. The disassembly of such programs might result in ambiguous results if the binary employs self modifying code techniques.

VI. *DYNAMIC MALWARE ANALYSIS*

A given malware sample can be executed within a controlled environment and monitoring its actions in order to analyze the malicious behavior which is called dynamic malware analysis. Since Dynamic Malware Analysis is performed during runtime and malware unpacks itself, dynamic malware analysis evades the restrictions of static analysis (i.e., unpacking issue). Thereby it is easy to see the actual behavior of a program. However, the main drawback is so-called dormant code: That is, unlike static analysis, dynamic analysis usually monitors only one execution path and thus suffers from incomplete code coverage. In addition there is the danger of harming third party systems, if the analysis environment is not properly isolated or restricted respectively. Furthermore, malware samples may alter their behavior or stop executing at all once they detect to be executed within a controlled analysis environment.

There are two basic approaches for dynamic malware analysis which are as below:

- *Analyzing the difference between defined points:* A given malware sample is executed for a certain period of time and afterwards the modifications made to the system are analyzed by comparison to the initial system state. In this approach, Comparison report states behavior of malware.
- *Observing runtime-behavior:* In this approach, malicious activities launched by the malicious application are monitored during runtime using a specialized tool

An example of first approach is Regshot tool. Before executing the binary, we will take a snapshot of the registry with Regshot. After executing the binary, we will take the second snapshot by clicking the 2nd shot button and then compare the two snapshots by clicking the compare button. When analysis is complete, we got result in text file such as which file are added and modified.

While observing the runtime-behavior of an application is currently the most promising approach. It is mostly conducted utilizing sandboxing. A sandbox hereby refers to a controlled runtime environment which is partitioned from the rest of the system in order to isolate the malicious process. This partitioning is typically achieved using virtualization mechanisms on a certain level.

VII. *MALWARE ANALYSIS TOOLS*

Here is an overview of the existing approaches and tools that make use of the presented techniques to analyze unknown and potentially malicious software. The analysis reports generated by the tools in this section give an analyst valuable insights into actions performed by a sample. These reports lay the foundation for a fast and detailed understanding of the sample.

- *FileMon*: The FileMon program is very useful in finding changes to the file system. Additionally, any searches performed by the binary will be detected and recorded. This tool is rather noisy and picks up hundreds of file changes by a seemingly idle Windows system. Therefore be sure to clear the tool prior to executing the binary, and "stop capture" about 10 seconds after launching the tool.

- *Norman Sandbox:* The Norman Sandbox is a dynamic malware analysis solution which executes the sample in a tightly-controlled virtual environment that simulates a Windows operating system. This environment is used to simulate a host computer as well as an attached local area network and, to some extent, Internet connectivity. The core idea behind the Norman Sandbox is to replace all functionality that is required by an analyzed sample with a simulated version thereof. The simulated system thus has to provide support for operating system relevant mechanisms such as memory protection and multi-threading support. Moreover, all required APIs have to be present to give the sample the fake impression that it is running on a real system. Because the malware is executed in a simulated system, packed or obfuscated executables do not hinder the analysis itself. Norman Sandbox focuses on the detection of worms that spread via email or P2P networks, as well as viruses that try to replicate over network shares.
- *JoeBox:* During the dynamic analysis of a potentially malicious sample, JoeBox creates a log that contains high level information of the performed actions regarding file system, registry, and system activities. JoeBox is specifically designed to run on real hardware, and not to rely on any virtualization or emulation technique. The system is designed as a client server model where a single controller instance can coordinate multiple clients that are responsible for performing the analysis. Thus, it is straight forward to increase the throughput of the complete system by adding more analyzing clients to the system. All analysis data is collected by the controlling machine.

VIII. *TRENDS IN MALWARE*

Malware is growing increasingly sophisticated. Malware authors seek to make their tools undetectable. Virtually every known offensive technique has been incorporated into malware to make it more difficult to defend against. Malware authors often seek to deliver several components in a single malware payload. Such additional components can include kernel level drivers designed to hide the presence of the malware, and malware client and server components to provide proxy services through an infected computer. One technique for embedding these additional components within Windows malware is to make use of the resource sections within Windows binaries. Malware may choose to create its own installation directory deep within the install program's hierarchy in an attempt to hide from curious users. Various techniques also exist to prevent installed antivirus programs from detecting a newly infected computer. A crude yet effective method is to modify a system's hosts file to add entries for hosts known to be associated with antivirus updates. A hosts file is a simple text file that contains mappings of IP address to hostnames. The modifications go so far as to insert a large number of carriage returns at the end of the existing host entries before appending the malicious host entries in the hopes that the casual observer will fail to scroll down and notice the appended entries. By causing antivirus updates to fail, new generations of malware can go undetected for long periods. Malware authors are increasingly turning to the use of rootkit techniques to hide the presence of their malware. Most malware takes steps to ensure that it will continue to run even after a system has been restarted. The most basic forms of persistence are achieved by adding commands to system start up scripts that cause the malware to execute. On Windows systems this evolved to making specific registry modifications to achieve the same effect. Other registry manipulations include installing malware components as extensions to commonly used software such as Windows Explorer or Microsoft Internet Explorer. More recently, malware has taken to installing itself as an operating system service or device driver so that components of the malware operate at the kernel level and are launched at system start up.

IX. DE-OBFUSCATING MALWARE

Obfuscation is the process of modifying something so as to hide its true purpose. In the case of malware, obfuscation is used to make automated analysis of the malware nearly impossible and to frustrate manual analysis to the maximum extent possible. There are two basic ways to deal with obfuscation. The first way is to simply ignore it, in which case your only real option for understanding the nature of a piece of malware is to observe its behaviour in a carefully instrumented environment. The second way to deal with obfuscation is to take steps to remove the obfuscation and reveal the original "de-obfuscated" program, which can then be analyzed using traditional tools such as disassembles and debuggers. Of course, malware authors understand that analysts will attempt to break through any obfuscation, and as a result they design their malware with features designed to make de-obfuscation difficult. De-obfuscation can never be made truly impossible since the malware must ultimately run on its target CPU; it will always be possible to observe the sequence of instructions that the malware executes using some combination of hardware and software tools. Tools used to obfuscate compiled binary programs are generically referred to as packers. This term stems from the fact that one technique for obfuscating a binary program is simply to compress the program, as compressed data tends to look far more random, and certainly does not resemble machine language. For the program to actually execute on the target computer, it must remain a valid executable for the target platform. The most basic packers simply perform compression of a binary's code and data sections. More sophisticated packers not only compress, but also perform some degree of encryption of the binary's sections. There are different tool available for unpacking binary.

- *Debugger-Assisted Unpacking:* Allowing malware to run free is not always a great idea. If we don't know what the malware does, it may have the opportunity to wreak havoc before we can successfully dump the memory image to disk. Debuggers offer greater control over the execution of any program under analysis. The basic idea when using a debugger is to allow the malware to execute just long enough for it to unpack itself, then to utilize the memory dumping capabilities of the debugger to dump the process image to a file for further analysis. A fundamental problem when working with self-modifying code in a debugger is that software breakpoints (such as the x86 int 3) are difficult to use since the saved breakpoint opcode (0xCC on the x86) may be modified before the program reaches the breakpoint location. As a result, the CPU will fetch something other than the breakpoint opcode and fail to break properly. Hardware breakpoints could be used on processors that support them; however, the problem of where to set the breakpoint remains. Without a correct disassembly, it is not possible to determine where to set a breakpoint. The only reasonable approach is to use single stepping until some pattern of execution such as a loop is revealed, then to utilize breakpoints to execute the loop to Completion, at which point you resume single stepping and repeat the process.

X. *CONCLUSION*

We have learnt Malware basics, malware analysis and techniques of analyzing malware. We have also learnt limitations of static malware analysis. After the discussion between static and malware analysis, Dynamic malware analysis is the best way to analyze malware samples. In this we have gone through the some tools for malware analysis.
We also see current trends in malware and de-obfuscating malware.

XI. REFERENCES

1) A Survey on Automated Dynamic Malware Analysis Techniques and Tools, http://www.seclab.tuwien.ac.at/papers/malware_survey.pdf
2) Malware Analysis & its Application to Digital Forensic, http://www.enggjournals.com/ijcse/doc/IJCSE12-04-04-023.pdf
3) Gray Hat Hacking 2nd Edition McGraw Hill by Shon Harris