Automated Malware Analysis Using Sandbox

Sumesh K J *[1],* P Rakesh *[2],* S Vijay Pavan *[2]*

***Abstract*—The Internet has become an essential part of the daily life of many people. More and more people are making use of services that are offered on the Internet. The Internet has evolved from a basic communication network to an interconnected set of information sources enabling, among other things, new forms of (social) interactions and market places for the sale of products and services. Online banking or advertising are mere examples of the commercial aspects of the Internet. Just as in the physical world, there are people on the Internet with malevolent intents that strive to enrich themselves by taking advantage of legitimate users whenever money is involved. Malware (i.e., software of malicious intent) helps these people accomplishing their goals.**

***Keywords*—Malware, Internet, threats, Network.**

I. Introduction

Malicious software, or malware, plays a part in most computer intrusion and security incidents. Any software that does something that causes harm to a user, computer, or network can be considered malware, including viruses, trojan horses, worms, rootkits, scareware, and spyware. While the various malware incarnations do all sorts of different things (as youll see throughout this book), as malware analysts, we have a core set of tools and techniques at our disposal for analyzing malware. Malware analysis is the art of dissecting malware to understand how it works, how to identify it, and how to defeat or eliminate it. And you dont need to be an uber-hacker to perform malware analysis. With millions of malicious programs in the wild, and more encountered every day, malware analysis is critical for anyone who responds to computer security incidents. And, with a shortage of malware analysis professionals, the skilled malware analyst is in serious demand. That said, this is not a book on how to find malware. Our focus is on how to analyze malware once it has been found. We focus on malware found on the Windows operating systemby far the most common operating system in use todaybut the skills you learn will serve you well when analyzing mal-ware on any operating system. We also focus on executables, since they are the most common and the most difficult files that youll encounter. At the same time, weve chosen to avoid discussing malicious scripts and Java programs. Instead, we dive deep into the methods used for dissecting advanced threats, such as backdoors, covert malware, and rootkits.

II. Malware Analysis

Today, signatures for anti-virus toolkits are created manually. Prior to writing a signature, an analyst must know if an unknown sample poses a threat to the users. Dierent malware analysis techniques allow the analyst to quickly and in detail understand the risk and intention of a given sample. This insight allows the analyst to react to new trends in malware development or rene existing detection techniques to mitigate the threat coming from that sample. The desire of analysts to understand the behavior of a given sample, and the opposing intention of malware authors, to disguise their creations malicious intents, leads to an arms race between those two parties. As analysis tools and techniques become more elaborate, attackers come up with evasion techniques to prevent their malware from being analyzed. Such techniques cover self modifying or dynamically generated code, as well as approaches that detect the presence of an instrumented analysis environment, thus, allowing the malware to conceal or inhibit its malicious behavior. Before we elaborate on possible evasion mechanisms, the following sections present an overview of applicable program analysis techniques that are used today to analyze malicious code. The process of analyzing a given program during execution is called dynamic analysis, while static analysis refers to all techniques that analyze a program by inspecting it.

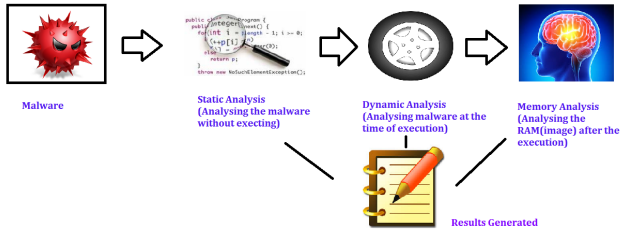

Fig. 1. Block Diagram of Analysis

III. Related Work

Examining Malicious software involves several stages, however the stages mentioned can be considered as discrete and sequential steps over-simplifies the steps malware analysis process. Manual Code Reversing, Interactive Behavior Analysis, Static Properties Analysis.All the above steps have to be done manually and the reports have to be generated manually.The reports have to be taken for further analysis.In this paper we were discussing about how to automate the whole three analysis which are static analysis, dynamic analysis, memory analysis. For completeness, a brief introduction and known limitations of static analysis approaches follows.

[1] Sumesh K J is with the Department of Computer Science and Engineering, Amrita University, Amritapuri, Kerala, 690525 India e-mail: (see sumeshkj@am.amrita.edu).

[2] P Rakesh and S Vijay Pavan are with Amrita University.

April , 2017.

A. Static Analysis

Static analysis consists of examining the executable file without viewing the actual instructions. Basic static analysis can confirm whether a file is malicious, provide information about its functionality, and sometimes provide information that will allow you to produce simple network signatures. Basic static analysis is straightforward and can be quick, but its largely ineffective against sophisticated malware, and it can miss important behaviors.

1) Antivirus Scanning- A Useful First Step: When first analyzing prospective malware, a good first step is to run it through multiple antivirus programs, which may already have identified it. But antivirus tools are certainly not perfect. They rely mainly on a database of identifiable pieces of known suspicious code (file signatures), as well as behavioral and pattern-matching analysis (heuristics) to identify suspect files. One problem is that malware writers can easily modify their code, thereby changing their programs signature and evading virus scanners. Also, rare malware often goes undetected by antivirus software because its simply not in the database. Finally, heuristics, while often successful in identifying unknown malicious code, can be bypassed by new and unique malware. Websites such as VirusTotal (http://www .virustotal.com/) allow you to upload a file for scanning by multiple antivirus engines. VirusTotal generates a report that provides the total number of engines that marked the file as malicious, the malware name, and, if available, additional information about the malware.

2) Hashing- A Fingerprint for Malware: Hashing is a common method used to uniquely identify malware. The malicious software is run through a hashing program that produces a unique hash that identifies that malware (a sort of fingerprint). The Message-Digest Algorithm 5 (MD5) hash function is the one most commonly used for malware analysis, though the Secure Hash Algorithm 1 (SHA-1) is also popular. For example, using the freely available md5deep program to calculate the hash of the Solitaire program that comes with Windows would generate the following output: C: $md5deepc$: 32.exe373$e7a$863$a1a$345c60edb9e20ec3231c : 32.$exeThehashis$373$e7a$863$a1a$345c60edb9e20ec3231.

3) Network Simulation: Modern malware samples frequently require some form of Internet access for their operation. For example, a malware sample might download additional components, updates, or conguration data before performing its nefarious actions. The trivial approach of denying or allowing all network traffic usually yields undesired results. Denying all network access to the sample under analysis will most likely result in incomplete observations of the malwares behavior. For example, a spam sending worm that cannot resolve the IP address of the mail-server cannot even try to send spam emails. Thus, this characteristic behavior cannot be monitored. If all network access is allowed, however, the same sample might participate in real spam campaigns which resembles unacceptable behavior. In addition to providing Internet access to a malware sample, it is also advantageous to provide

```
C:\Users\[redacted]\Desktop>volatility-2.3.1.standalone.exe -f image.dd connscan
Volatility Foundation Volatility Framework 2.3.1
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x05554008 127.0.0.1:1229            127.0.0.1:2226            472
0x055f5008 127.0.0.1:1416            127.0.0.1:2226            664
0x0571f008 127.0.0.1:2226            127.0.0.1:1229            1668
0x05721008 0.0.0.0:1498              24.177.33.91:2297         1240
0x05b77860 111.112.113.52:1075       216.163.188.45:80         3288
0x05c10e68 111.112.113.52:1025       111.112.113.199:445       4
0x05c1a008 111.112.113.52:1116       184.27.23.30:443          3696
0x05dbb860 111.112.113.52:1074       216.163.188.45:80         3288
0x071ca860 111.112.113.52:1074       216.163.188.45:80         3288
0x08c26008 127.0.0.1:1229            127.0.0.1:2226            472
0x094f0008 127.0.0.1:2226            127.0.0.1:1229            1668
0x12e83008 111.112.113.52:1116       184.27.23.30:443          3696
0x152d7e68 111.112.113.52:1025       111.112.113.199:445       4
0x1a589008 127.0.0.1:1416            127.0.0.1:2226            664
0x241b0008 127.0.0.1:1229            127.0.0.1:2226            472
```

Fig. 2. List of Tcp Connections

easy targets for infection in case the malware attempts to spread over the network. Such targets are commonly provided by honeypots that imitate vulnerable network services. In this section, we discuss methods to restrict network access for samples under analysis. These restrictions can be performed with dierent techniques and to varying extents. No Internet, but simulated network. This approach does not allow the sample under analysis to communicate with the real Internet. Instead, commonly used network services are simulated and all trac is redirected to these services. Such services include but are not limited to DNS, mail, IRC, and le servers. If the simulation is complete enough the malware will exhibit its malicious behavior and the analysis is completely self-contained. Malware that tries to update over the Internet will fail to do so, and bots that wait until they receive commands from their master most likely will stay dormant.

4) Finding Strings: A string in a program is a sequence of characters such as the. A program contains strings if it prints a message, connects to a URL, or copies a file to a specific location. Searching through the strings can be a simple way to get hints about the functionality of a program. For example, if the program accesses a URL, then you will see the URL accessed stored as a string in the program. You can use the Strings program (http://bit.ly/ic4plL), to search an executable for strings, which are typically stored in either ASCII or Unicode format.

5) Packed and Obfuscated Malware: Malware writers often use packing or obfuscation to make their files more difficult to detect or analyze. Obfuscated programs are ones whose execution the malware author has attempted to hide. Packed programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed. Both techniques will severely limit your attempts to statically analyze the malware. When the packed program is run, a small wrapper program also runs to decompress the packed file and then run the unpacked file.When a packed program is analyzed statically, only the small wrapper program can be dissected.

Detecting Packers with PEiD- One way to detect packed files is with the PEiD program. You can use PEiD to detect the type of packer or compiler employed to build an application, which makes analyzing the packed file much easier.As you can see, PEiD has identified the file as being packed with UPX version 0.89.6-1.02 or 1.05-2.90. (Just ignore the other information shown here for now.

6) Linked Libraries and Functions: One of the most useful pieces of information that we can gather about an executable is the list of functions that it imports. Imports are functions used by one program that are actually stored in a different program, such as code libraries that contain functionality common to many programs. Code libraries can be connected to the main executable by linking. Programmers link imports to their programs so that they dont need to re-implement certain functionality in multiple programs. Code libraries can be linked statically, at runtime, or dynamically. Knowing how the library code is linked is critical to our understanding of malware because the information we can find in the PE file header depends on how the library code has been linked.

B. Dynamic Analysis

Dynamic analysis techniques involve running the malware and observing its behavior on the system in order to remove the infection, produce effective signatures, or both. However, before you can run malware safely, you must set up an environment that will allow you to study the running malware without risk of damage to your system or network. Like basic static analysis techniques, basic dynamic analysis techniques can be used by most people without deep programming knowledge, but they wont be effective with all malware and can miss important functionality.

1) Using a Malware Sandbox: Many malware sandboxes-such as Norman SandBox, GFI Sandbox, Anubis, Joe Sandbox, ThreatExpert, BitBlaze, and Comodo Instant Malware Analysis will analyze malware for free. Currently, Norman SandBox and GFI Sandbox (formerly CWSandbox) are the most popular among computer-security professionals. These sandboxes provide easy-to-understand output and are great for initial triage, as long as you are willing to submit your malware to the sandbox websites. Even though the sandboxes are automated, you might choose not to submit malware that contains company information to a public website.

2) Running Malware: Basic dynamic analysis techniques will be rendered useless if you cant get the malware running. Here we focus on running the majority of malware you will encounter (EXEs and DLLs). Although youll usually find it simple enough to run executable malware by double-clicking the executable or running the file from the command line, it can be tricky to launch malicious DLLs because Windows doesnt know how to run them automatically. Lets take a look at how you can launch DLLs to be successful in performing dynamic analysis. The program rundll32.exe is included with all modern versions of Windows. It provides a container for running a DLL using this syntax: C: $rundll32.exeDLLname, Exportarguments$.

The Export value must be a function name or ordinal selected from the exported function table in the DLL.you can use a tool such as PEview or PE Explorer to view the Export table. For example, the file rip.dll has the following exports: Install Uninstall Install appears to be a likely way to launch rip.dll, so lets launch the mal-ware as follows: C: $rundll32.exerip.dll, Install$

Malware can also have functions that are exported by ordinalthat is, as an exported function with only an ordinal number, which we discussed in depth in Chapter 1. In this case, you can still call those functions with rundll32.exe using the following command, where 5 is the ordinal number that you want to call, prepended with the character:

C: $rundll32.exexyzzy.dll, 5$

Because malicious DLLs frequently run most of their code in DLLMain (called from the DLL entry point), and because DLLMain is executed whenever the DLL is loaded, you can often get information dynamically by forcing the DLL to load using rundll32.exe. Alternatively, you can even turn a DLL into an executable by modifying the PE header and changing its extension to force Windows to load the DLL as it would an executable.

To modify the PE header, wipe the IMAGEFILEDLL (0x2000) flag from the Characteristics field in the IMAGEFILEHEADER.

While this change wont run any imported functions, it will run the DLLMain method, and it may cause the mal-ware to crash or terminate unexpectedly.

However, as long as your changes cause the malware to execute its malicious payload, and you can collect information for your analysis, the rest doesnt matter. DLL malware may also need to be installed as a service, sometimes with a convenient export such as InstallService, as listed in ipr32x.dll:

C: $rundll32ipr32x.dll, InstallServiceServiceName$

C: $netstartServiceName$

3) Viewing Processes with Process Explorer: The Process Explorer, free from Microsoft, is an extremely powerful task manager that should be running when you are performing dynamic analysis. It can provide valuable insight into the processes currently running on a system. You can use Process Explorer to list active processes, DLLs loaded by a process, various process properties, and overall system information. You can also use it to kill a process, log out users, and launch and validate processes.Process Explorer monitors the processes running on a system and shows them in a tree structure that displays child and parent relationships. Process Explorer shows five columns: Process (the process name), PID (the process identifier), CPU (CPU usage), Description, and Company Name. The view updates every second. By default, services are highlighted in pink, processes in blue, new processes in green, and terminated processes in red. Green and red highlights are temporary, and are removed after the process has started or terminated. When analyzing malware, watch the Process Explorer window for changes or new processes, and be sure to investigate them thoroughly. Process Explorer can display quite a bit of information for each process. For example, when the DLL information display window is active, you can click a process to see all DLLs it loaded into memory. You can change the DLL display window to the Handles window, which shows all handles held by the process, including file handles, mutexes, events, and so on.

4) Faking a Network: Malware often beacons out and eventually communicates with a command-and-control server, as

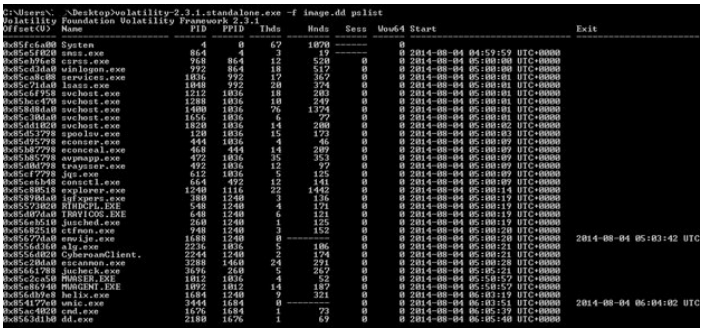

Fig. 3. List of all Process

well discuss in depth in Chapter 14. You can create a fake network and quickly obtain network indicators, without actually connecting to the Internet. These indicators can include DNS names, IP addresses, and packet signatures. To fake a network successfully, you must prevent the malware from realizing that it is executing in a virtualized environment. (See Chapter 2 for a discussion on setting up virtual networks with VMware.) By combining the tools discussed here with a solid virtual machine network setup, you will greatly increase your chances of success. ApateDNS spoofs DNS responses to a user-specified IP address by listening on UDP port 53 on the local machine. It responds to DNS requests with the DNS response set to an IP address you specify. ApateDNS can display the hexadecimal and ASCII results of all requests it receives.

5) Monitoring with Netcat: Netcat, the TCP/IP Swiss Army knife, can be used over both inbound and outbound connections for port scanning, tunneling, proxying, port forwarding, and much more. In listen mode, Netcat acts as a server, while in connect mode it acts as a client. Netcat takes data from standard input for transmission over the network. All the data it receives is output to the screen via standard output. Lets look at how you can use Netcat to analyze the malware RShell.Using ApateDNS, we redirect the DNS request for evil.malwar3.com to our local host. Assuming that the malware is going out over port 80 (a common choice), we can use Netcat to listen for connections before executing the malware.

6) Packet Sniffing with Wireshark: Wireshark is an open source sniffer, a packet capture tool that intercepts and logs network traffic. Wireshark provides visualization, packet-stream analysis, and in-depth analysis of individual packets. Like many tools discussed in this book, Wireshark can be used for both good and evil. It can be used to analyze internal networks and network usage, debug application issues, and study protocols in action. But it can also be used to sniff passwords, reverse-engineer network protocols, steal sensitive information, and listen in on the online chatter at your local coffee shop.Wireshark can help you to understand how malware is performing network communication by sniffing packets as the malware communicates. To use Wireshark for this purpose, connect to the Internet or simulate an Internet connection, and then start Wiresharks packet capture and run the malware.

C. Memory Analysis

Memory Forensics is the analysis of the memory image taken from the running computer. Memory forensics plays an important role in investigations and incident response. It can help in extracting forensics artifacts from a computers memory like running process, network connections, loaded modules etc. It can also help in unpacking, Rootkit detection and reverse engineering. When an organization is a victim of advanced malware infection, a quick response action is required to identify the indicators associated with that malware to remediate, establish better security controls and to prevent future ones from occurring. In this article you will learn to detect advance malware infection in memory using a technique called Memory Forensics and you will also learn to use Memory Forensic Toolkits such as Volatility to detect advanced malware with a real case scenario.

REFERENCES

Bayer, U., Moser, A., Krugel, C., and Kirda, E. 2006. Dynamic analysis of malicious code. Journal in Computer Virology 2, 1, 6777.

Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software By Michael Sikorski, Andrew Honig.

H. Carvey. Windows Forensics Analysis.Syngress, 2007.

B. M. An introduction to windows memory forensic. Technical report, July 2005. http://forensic.seccure.net/pdf/

M.Anderberg. Cluster Analysis for Applications.Academic Press,Inc.,NewYork,NY,USA, 1973.

P.Fogla, M.Sharif, R.Perdisci, O.Kolesnikov,andW.Lee.Polymorphic Blending Attacks. In Proceedings of USENIX Security Symposium, pages 241256, Vancouver, BC, Canada, 2006. USENIX Association.

Finding Advanced Malware Using Voltality by Monnappa Ka.

https://bruteforce.gr/yara-a-beginners-guide.html [online]

https://github.com/volatilityfoundation/volatility/

https://securingtomorrow.mcafee.com/mcafee-labs/overview-malwareself-defense-protection/ [online]

http://4tphi.net/fatkit/papers/NickMaclean2006.pdf