

Malware Analysis for Administrators

S. G. Masood 2004-05-20

1. Introduction

The threat of malicious software can easily be considered as the greatest threat to Internet security. Earlier, viruses were, more or less, the only form of malware. Nowadays, the threat has grown to include network-aware worms, trojans, DDoS agents, IRC Controlled bots, spyware, and so on. The infection vectors have also changed and grown and malicious agents now use techniques like email harvesting, browser exploits, operating system vulnerabilities, and P2P networks to spread. A relatively large percentage of the software that a normal internet user encounters in his online journeys is or can be malicious in some kind of way. Most of this malware is stopped by antivirus software, spyware removal tools and other similar tools. However, this protection is not always enough and there are times when a small, benign looking binary sneaks through all levels of protection and compromises user data. There may be many reasons for this breach, such as a user irregularly updating his AV signatures, a failure of AV heuristics, the introduction of new or low-profile malware which has not yet been discovered by AV vendors, and custom coded malware which cannot be detected by antivirus software. Though AV software is continually getting better, a small but very significant percentage of malware escapes the automated screening process and manages to enter and wreak havoc on networks. Unfortunately, this percentage is also growing everyday.

It is essential for users and absolutely essential for administrators to be able to determine if a binary is harmful by examining it manually and without relying on the automated scanning engines. The level of information desired differs according to the user's needs. For instance, a normal user might only want to know if a binary is malicious or not, while an administrator might want to completely reverse engineer the binary for his purposes.

Traditionally, malware analysis has been considered to be very complicated, and in fact some of the techniques are still very complicated and beyond a normal user's access. Nevertheless, looking at the current scenario, we can see that there is a clear need for people to learn how to analyze malware themselves. But the caveat is that the analysis techniques have to be simplified and the learning curve has to be made smaller for mass consumption among the general public. Unfortunately, there is not much organized information in the public domain dealing with easy to use malware analysis techniques. This paper tries to fill this void. The focus is on malware reversing but these techniques can be applied to reverse engineer any binary.

Besides the uses mentioned above, malware analysis is used for forensics, honeypot research, security vulnerability research, etc.

2. Background, goals, assumptions and tools

2.1 Background

There are basically two broad categories of techniques that are used for analyzing malware: code analysis and behaviour analysis. In most cases, a combination of both these techniques is used. We will consider code analysis first.

Code analysis is one of the primary techniques used for examining malware. The best way of understanding the way a program works is, of course, to study the source code of the program. However, the source code for most malware is not available. Malicious software is more often distributed in the form of binaries, and binary code can still be examined using debuggers and disassemblers. However, the use of these tools is often beyond the ability of all but a small minority because of the specialized knowledge required and the very steep learning curve needed to acquire it. Given sufficient time, any binary, however large or complicated, can be reversed completely by using code analysis techniques.

On the other hand, behaviour analysis is more concerned with the behavioural aspects of the malicious software. Like a beast kept under observation in a zoo, a binary can be kept in a tightly controlled lab environment and have its behaviour scrutinized. Things like changes it makes to the environment (file system, registry, network, etc), its communication with the rest of the network, its communication with remote devices, and so on are closely observed and information is collected. The collected data is analyzed and the complete picture is reconstructed from these different bits of information.

The best thing about behaviour analysis is that it is within the scope of an average administrator or even a power user. The learning curve is very small and existing knowledge can be leveraged to make the learning process faster. This makes it ideal for teaching newbies the art of malware reverse engineering. These reasons are consistent with our stated goals, focused on the typical administrator, and therefore this paper is mostly concerned with behaviour analysis.

Though reverse engineering using behaviour analysis does not lead to the complete reversing of

a binary, it is sufficient for most users' needs. For instance, it is not sufficient for an antivirus researcher but for most other users, behaviour analysis can fulfill all their needs.

2.2 Goals in the analysis

As stated before, our goal is to provide a set of behaviour analysis techniques for reverse engineering malware. Also, the learning curve should be small so that it is within the scope of most people.

Using these methods, people should be able to analyze an unknown binary and determine whether it is malicious or not. Those who require more in-depth knowledge should be able to reverse engineer the binary, understand and document its workings completely.

2.3 Assumptions and definitions

This paper makes a few assumptions for the sake of convenience and clarity. These are:

1. We assume that the malware under consideration is a Win32 based binary on an Intel x86 machine. This is just for the sake of clarity. The basic principles can be just as easily applied to any other platform.
2. We sometimes refer to the malware as "the binary". This does not however mean that the principles are applicable only to a malicious application that is composed of a single binary.
3. The host machine on which the binary is executed is referred to as the "victim host" or the "victim machine".
4. The other machine on the test network is referred to as the "sniffer machine".

2.4 Tools

Since the goal of this paper is to propose a generic set of techniques, the tools mentioned in this paper are just "proposed" tools and are available as references at the end of this document. Any other tool that has the same or similar functionality can be used in place of the proposed ones.

3. Methodology

The framework proposed is broadly divided into six stages. They are:

1. Creating a controlled environment
2. Baselining the environment
3. Information collection
4. Information analysis
5. Reconstructing the big picture
6. Documenting the results

3.1 Creating a controlled environment

The setting up of a controlled and sanitized environment is absolutely essential for analyzing malware. A special "test lab" is created for this purpose. Some essential features of the test lab are:

- At least two machines should be used. One machine is for hosting the malicious binary (victim machine) and the other is for baselining and sniffing the network traffic (sniffer machine). They should be networked in such a way that each of them is able to sniff the other's network traffic.
- The two networked lab machines should be isolated from the rest of the network.
- Fresh copies of Operating Systems should be installed on each of the two machines. It is preferable to have a WinNT kernel family OS on one machine and a *nix based OS on the other. Since we are assuming a Win32 binary, the WinNT machine acts as the "victim host" and the *nix machine is used as the "sniffer machine".
- Tools should be transferred to the relevant machines.
- The binary that is to be examined should be transferred to the relevant machine. Since we are assuming a Win32 binary, it is transferred to the Win32 machine in this case.
- It is highly preferable not to install any other application upon the "victim host" apart from the tools required for analysis.

This is the most basic setup for a malware analysis lab. Apart from this and depending on the situation, more modifications can be carried out. For instance, if the malicious binary tries to communicate with a remote server xyz.com, a DNS server has to be setup in one of the lab machines and a DNS entry for xyz.com has to be created. An excellent paper that discusses the creation of a malware analysis lab is "[An Environment for Controlled Worm Replication and Analysis](#)".

We may have to return to this "creating a controlled environment" stage many times during the

analysis process. Sometimes, in the light of new information generated during the later stages, the lab will have to be tweaked and modified.

3.2 Baselining the environment

Baselining the environment is the next major step. "Baselining" means taking a snapshot of the current environment. This is the most vital stage in our analysis. If baselining is not done properly, it has a serious effect on the information gathering stage, which in turn seriously affects our understanding of the binary. If baselining is done efficiently, the information generated during the next stage becomes very accurate and the rest of the stages become easy to execute.

To accomplish our goals, the binary which is to be analyzed is executed in a controlled environment and the changes it makes to that environment are captured. Before executing the binary, a snapshot of the environment is created (baseline) and then after execution another snapshot is created. In theory, the difference between the baseline and the final snapshot gives the changes made by the binary.

The elements of the environment that have to be baselined are:

3.2.1 Victim machine

Some of the elements that are to be baselined in the Victim Machine are:

- Filesystem: The file system on the victim host has to be baselined. There are many programs that can create a snapshot of the file system and after a few changes occur, they can point out the modifications. Some of the programs we can use are [Winalysis](#) and [Installrite](#).
- Registry: The registry is the next component that is to be baselined. Most malware applications rely on registry entries. Therefore it is crucial to capture registry modifications. Winalysis as mentioned above is one of the available programs that can be used for registry baselining.
- Running processes: A snapshot of the running processes can be created using a number of programs. Some of them are available from [Sysinternals](#).
- Open Ports: A snapshot of the open ports can be created using the 'netstat' utility. However, it does not list the name of the process that is tied to the port. For this, we can use [Fport](#) available from Foundstone.

- Users, Groups, Network Shares and Services are some of the other elements that should be baselined.

3.2.2 Network traffic

The next element that has to be baselined is the network traffic. Even when there is no application running on either of the test machines, there will still be some network traffic. This traffic has to be recorded and the "normal traffic" in our test network has to be defined. This is because when deviations occur in the "normal traffic" pattern, we can assume it to be generated by the binary and perform further testing on it.

Sniffing software that is installed on our "sniffer machine" is used for this purpose. Any sniffing software running in verbose mode is sufficient for our purposes. However, to make our task easier, it is preferable to use a protocol analyzer like [Ethereal](#).

3.2.3 External view

Although we have created a snapshot of the open ports in the victim machine, it is always better to create one more snapshot from an external machine. A port scanner running on our "sniffer machine" can achieve this task for us. It goes without saying that [Nmap](#) will be the port scanner of choice for most users.

3.3 Information collection

Now that the preparations are over, we can go ahead with our task. This is the only stage where we have an actual interaction with the binary. A lot of raw information about the binary is collected during this stage which is analyzed in the next stage. Therefore, it is very important to carefully record all the information generated in this stage. The steps in the information collection stage are:

3.3.1 Static analysis

During the static analysis stage, we collect as much information about the binary as possible, without executing it. This involves many techniques and tools. Static analysis reveals the scripts, HTML, GUI, passwords, commands, control channels, and so on. Simple things like the file name, size, version string (right-click>properties>version in Win32), are recorded.

Human-readable strings are extracted from the binary and these strings are recorded. A program like [Binary Text Scan](#) can be used for this purpose. These strings reveal a lot of information about the function of the binary.

Resources that are embedded in the binary are extracted and recorded. A program like [Resource Hacker](#) can be used for this purpose. The resources that can be discovered through this process include GUI elements, scripts, HTML, graphics, icons, and more.

3.3.2 Dynamic analysis

During this stage, we actually execute the binary and observe its interaction with the environment. All monitoring tools including the sniffing software are activated. Different experiments are done to test the response of the running malware process to our probes. Attempts to communicate with other machines are recorded. Basically a new snapshot of the environment is created like in the baselining the environment stage.

After taking a snapshot of all the changes the binary performs in the system, the binary process is terminated. Now, the differences between the new snapshot and the baseline snapshot are determined. The dynamic analysis step is very similar to the baselining the environment stage. Therefore, the tools are reused for this stage. Winalysis and InstallRite can be used for this purpose. Apart from these tools, Filemon and Regmon from [Sysinternals](#) can be used for monitoring the file system and the registry dynamically. These tools are used for observing the changes to the file system and the registry.

This information is recorded and forms the input for the next stage of our analysis. The information generated here can be new files, registry entries, open ports, etc.

Sometimes, the static analysis step has to be repeated once more after doing a dynamic analysis.

3.4 Information analysis

This is the stage where we can finally reverse engineer the binary based on all the information collected during the previous stages. Each part of the information is analyzed over and over and the "jigsaw puzzle" is completed. Then the big picture automatically begins to appear and the

reverse engineering process is finished. However, before this is achieved, we may have to repeat the previous stages (See figure) several times.

The goals of the individual or organization evaluating the binary determine the type of analysis and because the goals differ, no standard methodology is provided for this stage. Looking for deviations from the stated security policy of an organization based on the information can be the determining factor in some cases.

Although a complete methodology for information analysis is beyond the scope of this paper, a few techniques are presented here. In many cases, these techniques are sufficient for analysis.

3.4.1 Internet searches

A search engine can be used for searching for more information on the binary. Keywords for the search engine can be drawn from the information generated during the "Static Analysis" step during the previous stage. Things like filenames, registry entries, commands, etc. often reveal a lot of information about the malware. Some good sources of information on the internet include Online Virus Databases (mostly maintained by Antivirus Vendors), News Groups and Mailing Lists. Many times, internet searches reveal almost all there is to know about the malware and no further research is needed.

3.4.2 Startup methods

Every malware needs a way to ensure that it is executed when a system reboots. This is the most vulnerable aspect of the malware. There are only a limited number of ways in all operating systems that a program can use to restart automatically when a machine reboots. The information collected during the previous stage can be analyzed to identify the startup method of the malware. A very good source for Startup Methods related information on the Internet is the [Paul Collins' Startup List](#).

3.4.3 Communication protocol

A network protocol analyzer like Ethereal in many cases can identify the communication protocol used by the binary. When this is not the case, the protocol has to be reverse engineered. This is beyond the scope of this document.

3.4.4 Spreading mechanism

If the malware under scrutiny is a self-spreading worm or virus, the collected network traffic data will easily reveal its spreading mechanism. In most cases, a cursory glance is enough.

3.5 Documenting the results

Documenting the results of the malware analysis and reverse engineering exercise is essential. One of the main advantages is that the knowledge incorporated into the documentation can be leveraged for later analysis exercises. The documentation needs differ from individual to individual and organization to organization. The method preferred by the concerned party can be used here.

4. Conclusion

From this article we've seen that a basic behavioral analysis of a binary can be easily performed by an administrator, or indeed by a power user. While this approach does not give the same level of detail as code analysis would, it is sufficient for most people's needs when figuring out what a potentially malicious binary is capable of.

About the author

[S.G.Masood](#) is the founding CTO of the Chicago, Illinois based application security startup Circle Technologies. He currently stays in Hyderabad, India and manages the development center.

References

"An Environment for Controlled Worm Replication and Analysis" by Ian Whalley Bill Arnold, David Chess, John Morar, Alla Segal, Morton Swimmer - www.research.ibm.com/antivirus/SciPapers/VB2000INW.htm

"Reverse Engineering Malware" by Lenny Zeltser - www.zeltser.com/sans/gcih-practical/revmalw.html

"Paul Collins' Startup List" - <http://www.sysinfo.org/startuplist.php>

Archives of the various security and malware related [mailing lists](#), most notably, Bugtraq, Full-Disclosure, Focus-Virus, Incidents.

VMWare - www.vmware.com

Winalysis - www.winalysis.com

Installrite - www.epsilonssquared.com

Fport - www.foundstone.com

Nmap - www.insecure.org

Binary Text Scan - netninja.com/files/bintxtscan.zip

Resource Hacker - www.users.on.net/johnson/resourcehacker/

Filemon and Regmon - www.sysinternals.com

Ethereal - www.ethereal.com

Comments or reprint requests for this article can be sent to the [editor](#).

[Privacy Statement](#)

Copyright 2006, SecurityFocus