JASON CARPENTER

Analyzing Malware

Difficulty

This article is an introduction to analyzing malware. I will take you through the basic steps you need to perform in order to understand what malware is doing to your systems.

Malware is software designed to infiltrate or damage a computer system without the owner's informed consent. The expression is a general term meaning a variety of forms of hostile, intrusive, or annoying software or program code.

Simply put, Malware is software designed to make a computer do something an attacker wants it to do. It is not always designed to destroy a computer. It may, for example, just sit on a computer, using processor cycles to crack the encryption of a certain file.

Nowadays, Malware has become so prevalent in our computer systems that most people do not take it seriously. Malware infects the average user at least once, yet we continue to operate the recently infected machine to perform personal confidential transactions, such as online banking or shopping.

Malware poses a serious threat to an enterprise and can do anything the attacker can envision. It can use system resources such as CPU cycles or bandwidth, or it can send official and confidential corporate data offsite to the attacker. Most corporations have antivirus systems in place, and some even have antispyware capabilities.

However, most of the time corporations use these systems to prevent or clean up infections after their machines are compromised. Most organizations do not take the time to recognize and understand the extent in which malware has inflicted their systems before attempting to eliminate it. Unfortunately, being infected with malware is usually much easier than getting rid of it, and once you have malware on your computer it tends to multiply.

Determining how a malware is constructed and operates in order to study its potential to inflict damage is called analyzing malware.

Analyzing malware is beneficial to the enterprise. Most malware detection systems, such as an antivirus protection system, require signature files that match the malware in order to enable them to detect and block the malware from penetrating your machine. When a new malware hits the net, you are virtually unprotected since your antivirus or antispyware software does not contain the identifying signature of the new malware.

For a new malware to be detected there is often a time delay until the new signature is distributed, since anti-malware companies need to identify it, analyze it, find a signature, test the signature and deploy the new updates.

If you have already been infected, the time involved is unacceptable, especially if you have no idea that you are infected and/or the extent of damage.

An example of this would be an online shopping site. If a new malware hits the net, and it takes two weeks for your antivirus vendor to deploy a signature file, your site is exposed and entirely susceptible to the infection.

WHAT YOU WILL LEARN...

Why analyzing malware is important

How you should get started

WHAT YOU SHOULD KNOW...

The Basics of X86 assembly language, logical thinking and a clear understanding of how software works

Another example of the benefit to reversing malware is if your anti-malware system succeeds in detecting the malware as an infection, but in reality, it has only detected the malware's clone.

This clone may appear to be the same malware that the anti-malware suite thinks it is, but part of it has been programmed to do something different and new. For example, without analyzing the malware, you would have no idea that in addition to spreading via file sharing, it will also spread via email.

Tools

There are many different tools available that will help you analyze malware. Some of these tools were designed specifically for debugging, and analyzing software, others are designed to better understand your system. There are more tools available than there is time to learn them all. I have a core set of tools I always use, and other tools I use only if necessary.

I highly recommend you to try as many different tools as possible until you find the tools that suite your methods best.

I like to split tools into two groups, system orientated and software analyzing tools.

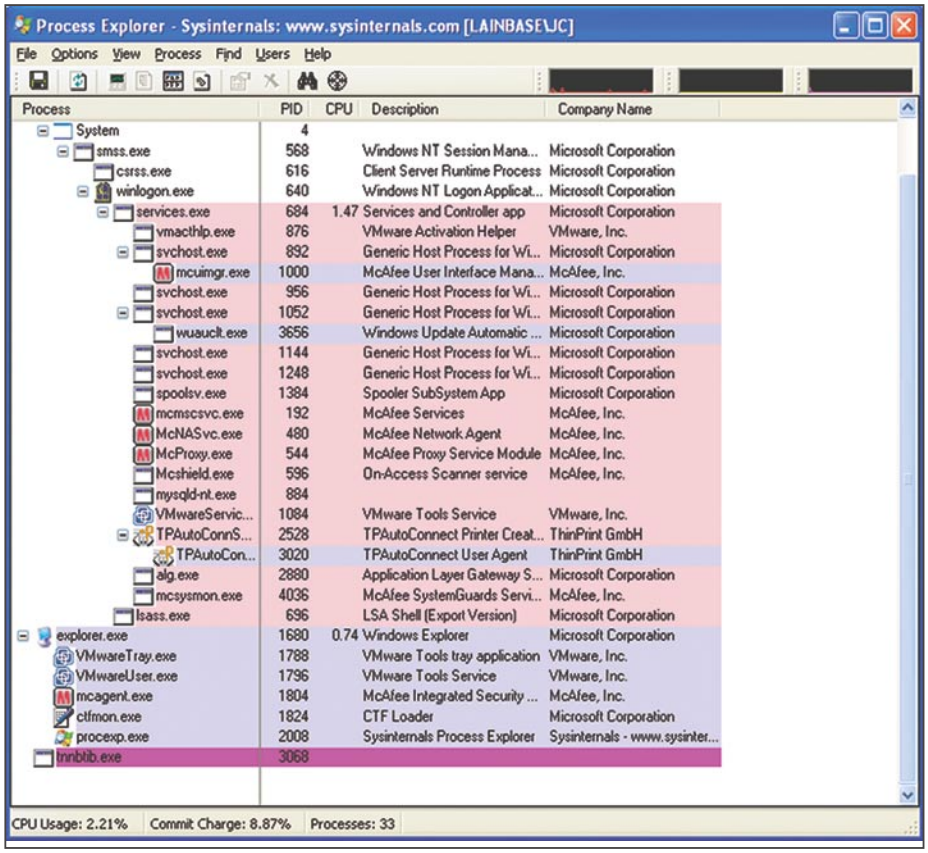

Figure 1. *Process Explorer showing processes running on the System*

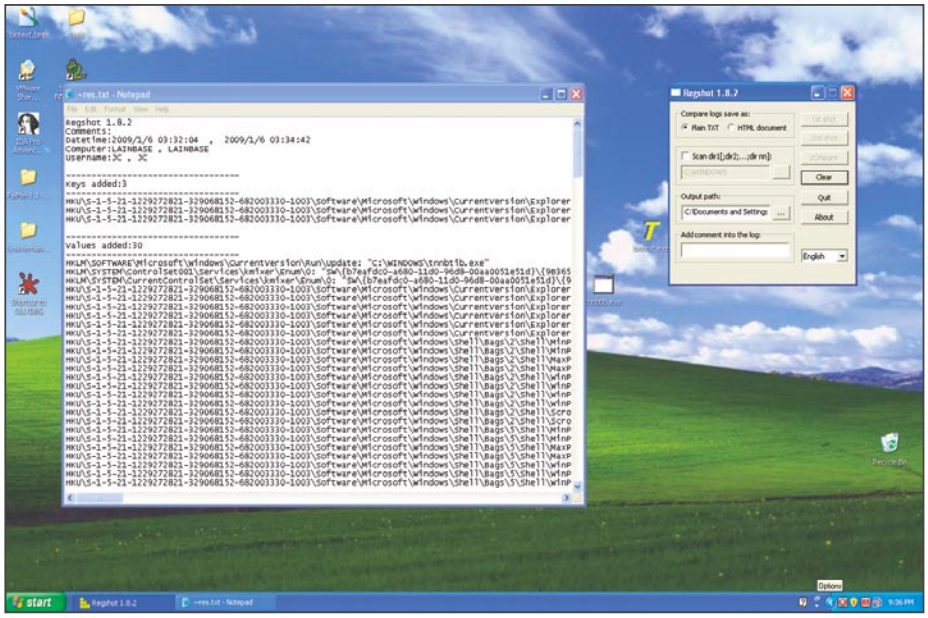

Figure 2. *Result of Regshot. This is comparing the registry before and after infection*

System Orientated

System orientated tools are tools designed to help you better understand your operating environment. Malware alters the system environment adding registry keys, files and network traffic.

It's difficult to recognize everything malware has done if you do not have a baseline with respect to your environment. These tools are most effective if you use them before and after an infection.

Microsoft Sysinternals

Microsoft's Sysinternals suite is one of the best tools out there to understand your Windows environment. It includes tools such as TCPView, Process explorer, and Autoruns.

This suite includes tools that let you see what processes are running, what ports are open, what files are set to run at startup amongst other things.

This really is a suite of tools that everyone should check out. I am not going to explain all these tools, as the best way to learn them is to use them on your system.

Regshot

One of the more challenging aspects of Microsoft Windows is the registry. New software often drops keys all over the place in the registry, but is too lazy to remove them upon uninstallation of the software. This makes the registry quite a mess.

This software is useful to help remove all the added keys a software installs. Just run it, install the software, run it again and it will show you the keys added or changed. Then when you remove the software you can verify it cleaned all the

keys out. This is also good in helping you determine what keys malware may have installed.

Snort

Snort is an open source intrusion detection system. It's a very useful software in any organization. It allows you to see actual traffic and analyze it to determine an attack on the network, unauthorized traffic, or who is hogging the bandwidth. It's really effective for looking at malware because it can log to a file that can be searched using grep and regular expressions.

NetCat

Netcat is a powerful open source TCP/IP tool. It can run a server, setup a tunnel, or a hexdump. Similar to the Unix command grep, it can take a while to fully understand all the uses of the tool, but once you get the hang of it, you will use it every chance you get. For malware, this tool is useful because it can help you see what happens when the malware makes a network connection.

Software Analyzing

While system orientated tools help you understand the environment, software analyzing tools are designed to help you understand the software itself. Software analyzing tools require a bit more in depth understanding of code then system orientated tools.

They require you understand programming methods, and low-level languages such as X86 Assembly. I will include some further resources on X86 Assembly in the reference section. The main type of software analyzing tools you will use is a debugger.

Debuggers

These tools are used to analyze binaries. Often used by software developers to find bugs in their code, these tools are the main tool used to analyze malware. Binaries are compiled code, designed to run on a system. Reversing binaries back to code is difficult as when they are compiled, they are stripped of non-essential information and optimized for processing.

Therefore when you use a debugger, the software usually can only report back instructions in a low-level format.

Some debuggers attempt to estimate what the original code may have looked like based on probability, these attempts are often fraught with mistakes. If you want to analyze malware I highly recommend you leave the code in low-level format.

IDA Pro

IDA Pro is one of the most used debuggers in the world. It has so many features that there are classes on the software, as well as books on how to use it. If you can swing the $515 to get the standard license or $985 for the advanced license, I highly recommend it. It will take time to learn thoroughly.

OllyDbg

If you like open-source and free, then OllyDbg is the way to go. It lacks some of the niceties of IDA pro, but it is efficient and has many plugins available to extend its usefulness.

Setting up the Environment

In order to reverse a malware you first need to setup a lab. I usually set it up using virtual machines. There are some malware however, that recognizes the VM and refuses to run. Therefore it helps to have some physical machines available as well, or alternatively a VM software that the malware will not recognize.

Virtual Machines provide us with the ability to roll back a host to a snapshot of an earlier time. This allows you to infect a machine, see how it works, and roll it back to a pristine state without having to rebuild the machine.

Whatever you decide to use, make sure that the lab environment is not connected to any other network. The last thing you want to do is allow the infection to spread to your own network, or to the Internet.

I find that in a lab, it helps to have more than one type of machine. The malware may infect more than one type of machine, and it may do that in a different way. For example, depending on the operating system, the malicious software could drop a file in either c:\windows or c:\winnt.

This is a simple example but you can see how malware can adapt based on the operating system. I also like to include different operating systems such as Linux along with Microsoft Windows because it allows me to have a wider array of tools at my disposal. I can create website in apache, or IIS, as well as use open source tools that are available only in Linux.

Malware Reversing Example

There are different ways to analyze malware. The two most common ways are behavioral analysis and code analysis. I prefer to do both, as it is more thorough.

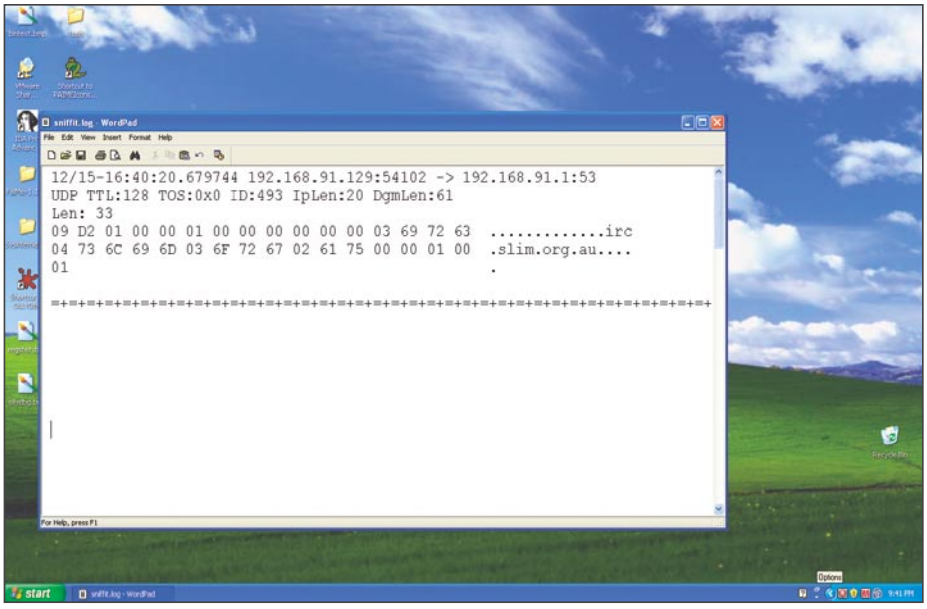

Figure 3. *A packet on the wire seen by snort. Notice the IRC server*

I will start with a behavioral analysis first. Essentially, I will watch the malware in action to see what it does. After that, I will do a code analysis to confirm my observations and findings, and look for any other actions the malware may perform that I did not recognize during the two analyses.

Behavioral Analysis

Let's go through an analysis together. First, we will setup a lab environment using virtual machines. These machines will include a windows XP machine that is the victim machine, and a Linux machine that we will use to check network traffic, act as a remote server, or act as another device on the network.

After I setup the lab, I need to determine what the generic Windows XP machine looks like before the infection. In order to do this, I will run parts of the Sysinternals suite provided by Microsoft. I will run the *Process Monitor* and *Process Explorer* tools. This will let me gain an insight regarding what is currently running on the system.

I will also use a tool called *Regshot* to take a baseline image of the registry. In order to determine what the malware attempts to do across the network I will use *TCPView*.

This tool shows me what connections are being established to and from my computer. Once I have a good understanding of the machine, I will infect a virtual machine and watch *Process Monitor*, *Process Explorer*, and TCPView to determine its effects.

I will also take another image using *Regshot* to determine what keys the malware has changed.

Running these tools I am able to determine a few things. First, using *Process Explorer*, I discover the malware started a process called *tnnbtib.exe* (See Figure 1).

Then, using *Regshot*, I was able to determine it also created a new file under c:\windows (a copy of itself) as well as new registry keys that pointed to that file in order to start it at runtime (Figure 2).

Using *TCPView*, I could also see it attempted to do a DNS resolution to an IRC channel and a web server. This is a good start, but it leads me to further questions such as *what does it do at the IRC channel*, or *why would it attempt to connect to a web server?* To investigate my questions and find further information, I decided to run Snort on the Linux virtual machine we setup earlier.

We set up this Linux virtual machine because it is always nice to have a box where you can run administrative commands and servers. This will let you determine what the malware will do if it tries to communicate with a remote server.

You do not have to use Linux, however I find that it is less expensive to run open source tools, and Linux has more tools available.

I run Snort first on my Linux virtual machine to monitor the network traffic generated by the malware. To run Snort I will use the command:

```
snort -vd | tee /tmp/sniffit.log
```

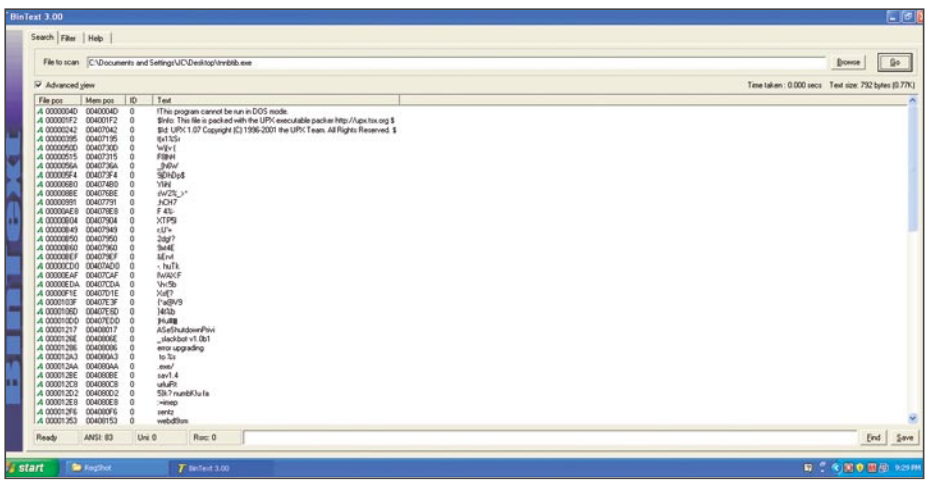

Figure 4. *BinText ran against the infection. You can see it was packed because the text is garbled. Also if you notice the type of packer is not garbled..UPX*

Then I will run the malware and watch the traffic. I can now see the DNS attempts to an IRC Channel and a web server (Figure 3).

My next step will be to configure the Windows machine to resolve those DNS entries to the Linux box. I do this by configuring the host file on Windows to resolve to the Linux box.

Then I setup an IRC server on the Linux box running on port 6666. This allows the malware to join its own channel. The malware does this by connecting with a random nickname.

After joining the IRC channel, the malware attempts to connect to a website. Curious as to what it is doing there, I setup *Netcat* to listen to port 80 with the command:

```
nc -p 80 -l -n
```

Netcat is a great tool to observe what traffic comes into a port; it is faster than setting up a web server and can be used for any port such as telnet or https as well. It is limited in that it will accept the packets, but since it is not a web server, it does not know how to respond and will dump them.

I was able to determine that it started a directory transversal as soon as it connected to that port. At this point, I felt I had a good idea about what this malware does, but I wanted to move into the next step, the code analysis.

So far we have determined that the malware created a file called bnntib.exe under the windows directory. It generated registry keys to start this file at boot.

Then it attempts to find an IRC channel. After connecting to the IRC channel with a random nickame it attempts to access a website and perform a directory transversal.

Our next step is to delve into the code and get a deeper understanding of the malware.

On the 'Net

Good assembly references

- An overview – *http://en.wikibooks.org/wiki/X86_Assembly.*
- Tutorials – *http://www.skynet.ie/~darkstar/assembler/.*
- This page discusses different assemblers and where to start – *http://webster.cs.ucr.edu/AsmTools/WhichAsm.html.*

Where to get tools

- Microsoft SysInternals – *http://technet.microsoft.com/en-us/sysinternals/default.aspx.*
- RegShot – *http://sourceforge.net/projects/regshot.*
- Snort – *http://www.snort.org/.*
- NetCat – *http://netcat.sourceforge.net/.*
- IDA Pro – *http://www.hex-rays.com/idapro/.*
- *http://www.ollydbg.de/.*
- BinText – *http://www.foundstone.com/us/resources/proddesc/bintext.htm.*

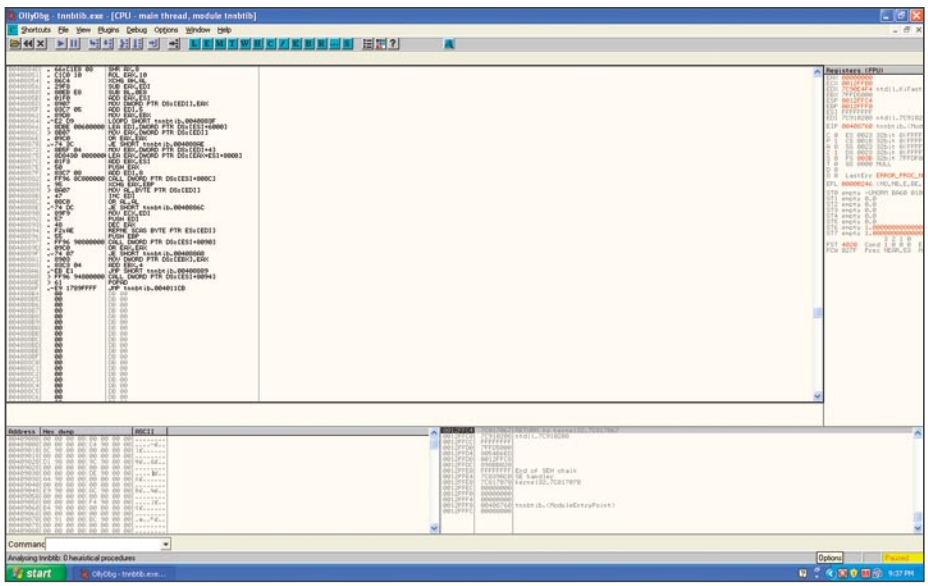

Figure 5. *Ollydbg analysis of the file*

Code Analysis

In order to do a code analysis of the malware, we first need to understand the difference between static and dynamic code analysis.

In static code analysis, the code is displayed but the file is not executed. *IDA Pro* is a good tool for performing this. It is useful because the code is not running and you can hop around the code as is, without taking up resources or running the malware. While performing static code analysis we must bear in mind a drawback. Since the code is not running, some of the calls to outside libraries are unavailable, together with the virtual memory address it would call.

Dynamic analysis tools, such as *OllyDbg*, actually step through the running code. This allows you to see everything it calls such as dynamic link libraries. I prefer to use dynamic analysis tools whenever possible because often malware uses packers and polymorphic code to conceal its code.

The Malware must unpack the code into memory first in order to execute it. Dynamic analysis tools are better at dealing with code that is dynamically loaded into memory.

"It helps to understand different methods malware authors use to defend against their malware being reversed.

These include packers and polymorphic code. Packers compress the file to a smaller size, a useful side effect of this is it makes the code difficult to read.

An example of this is UPX; in fact, UPX is very common amongst malware authors. Polymorphic code is code that changes while it runs.

This can make things difficult for static reversing as the code you are looking at in a static analyzer is not necessarily what the code will look like when it is actually running.

We will discuss more in depth parts of malware reversing, including PE tools, and

anti-debugging methods such as packing, or morphing code in part two of this article."

To continue our analysis, the first thing we are going to do is run *Bintext* and search for strings that will help us recognize the program. Examples would be open, close, connect etc...

Step 1. Bintext

However, in our example, most of the strings are illegible. We do see the words UPX. UPX is a common type of packer. This software extracts packed code into memory and runs it as if it was never packed. If you can determine the packer (UPX) you can often get the software and try to unpack it. This does not often work with malware.

There are several other ways to look at the code. You can run *PeID* to see if it recognizes it or use a dynamic debugger. Here in *OllyDbg*, I have located the instruction where the executable is already unpacked into memory (See Figure 4 and Figure 5).

By setting a break point here, we can run the program up to the breakpoint, step into the code and dump the debugged program from memory to disk. OllyDbg gives us the option to edit the headers or take the defaults *OllyDbg* figured out. With some of the packers we need to rewrite the headers. With this one I was able to take *OllyDbg's* defaults. After I saved it locally, I opened the unpacked code.

Digging deeper into the code we can recognize certain strings such as pass_ accepted, telling us there is an authentication system, and commands such as !@*upgrade* or !@*login*. By going back to our IRC server on Linux we can interact with the program by sending these commands such as !@*login* and the password karma. I found the password by supplying a bad one, setting up the strcmp call with a break point and when the bad password was compared against the good one, I could see both passwords on the stack.

Conclusion

After looking at this malware, I did not find any way for it to self propagate like a worm, or contain any useful program such as a Trojan.

Therefore, I determined it was probably a virus since it needed the user's intervention to run in order for the infection to spread. When this malware infects a computer, it drops a file into c:\windows, adds a key to the registry to run a process at boot time.

This virus was compressed using UPX. It connects to IRC and attempts to connect to a web server. It accepts commands that require authentication.

More than likely the author designed it to be part of a *botnet*, as it would allow a remote user to run commands over IRC.

Through the website traversal, it probably was going to pull down a file, perhaps something the attacker wants to crack by employing local resources.

The last thing a company ever wants is an attacker that controls their machines. This malware adds the machine to a *botnet* and allows the attacker to pull files from a website.

If this malware had a propagation method similar to a worm, we could have determined the need to inspect other machines.

This is a good example of why security teams should do more than just count on their anti-malware suites to clean the infection.

They need to understand the impact of malware on their organization. In part 2 of this article we will go further in depth on PE headers, and anti-reversing techniques such as anti-debuggers and polymorphic code.

Jason Carpenter
I have been in IT for 10 years now, doing everything from programming to administering networks. I am currently completing my master's degree in Information Assurance.