Fall 2018

Creating a Malware Analysis Lab and Basic Malware Analysis

Joseph Peppers
Iowa State University

Creating a Malware Analysis Lab and Basic Malware Analysis

by

Joseph Peppers

A creative component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTERS OF SCIENCE

Major: Information Assurance

Program of Study Committee:
Dr. Doug Jacobson, Major Professor

Iowa State University

Ames, Iowa

2018

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

AV	Anti-Virus
C&C	Command & Control
DDoS	Distributed Denial of Service
DLL	Dynamic Link Library
DNS	Domain Name System
GB	Gigabyte
GUI	Graphical User Interface
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
IDA	Interactive Disassembler
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
PC	Personal Computer
PE	Very Important Person
PID	Process Identification
SDLC	Software Development Life Cycle
URL	Uniform Resource Locator
VM	Virtual Machine

ABSTRACT

In tying together information learned in the Information Assurance program at Iowa State this paper goes over an introduction to malware, basic malware analysis, and setting up a manual malware analysis lab. Malware is malicious software that causes harm. The average malware will have 125 lines of code. Generally, malware consists of 3 components: a concealer, a replicator, and a bomb. Malware is classified based on its nature and functionality. The 3 most common we see are viruses, worms, and Trojans. Malware generally falls into two categories based on its target: mass malware and targeted malware. Four general stages of malware analysis are manual code reversing, interactive behavior analysis, static properties analysis, and automated analysis.

The paper goes over basic static and basic dynamic analysis. It briefly touches on advanced static and advanced dynamic analysis to cover 3 of the stages above. Sandboxes are covered and Cuckoo is talked about to cover automated analysis.

Setting up a malware analysis lab is talked about as a physical lab or a virtual lab can be set up. Steps are given to use VMWare Workstation Pro to set up a manual malware analysis lab, getting a Microsoft Windows virtual machine, and installing Fireeye’s flare-vm on it.

In closing, some work that can be expanded on and done in the future is discussed.

CHAPTER 1. INTRODUCTION: DEFINE MALWARE

The in studying malware, one of the most important first steps is understanding what malware is, what types are there, and how we can go about defining and relating malware. A short definition is malware is malicious software. A better version is “any software that does something that can causes harm to a user, computer, or network can be considered malware.” (Sikorski). The only piece I would tack on to that definition is with harm I would expand the definition to include having an adverse effect on a computer ecosystem by purposely using resources that it is not intended to. This would only be added as in today’s world there is malware going around that will try to install itself on various systems in order to crypto mine. Something like that infecting a company wouldn’t cause harm to a specific person, but could raise the cost of the servers through electricity, cooling costs, and CPU usage. Long term, it will likely have higher wear and tear on hardware as well.

The next big question to answer about malware is how big is it? From quite a few studies done around 2005 up to 2010 seem to have their binaries of 125 lines of code. Size wise, from Sophos’ Naked Security blog they list “In January 2005 the average size of a malware sample was 126 kB. In June 2010 it is 338 kB.” Stuxnet by comparison had close to 15,000 lines of code. Sikorski’s book states that the GNOME text editor is built on gedit.c and all its files taken into count on its base version is over 70,000 lines of code. The reason that we bring this up is to point out that in comparison to most software we have and use, finding malware code is almost the digital equivalent of looking for a needle in a haystack; it is generally going to be vastly outnumbered by non-malicious code.

120:1 Stuxnet to average malware
300:1 Simple Text Editor to average malware
2,000:1 Malware suite to average malware
100,000:1 Defensive tool to average malware
1,000,000:1 Target OS to average malware

Figure 1.1 *Found in Sikorski's book in the forward on page xxii summarizing lines of code in average malware versus various other pieces of code and software.*

The next main question to answer is what are we looking for exactly and why analyze malware? The answer to this is a bit more anecdotal than we would typically like, but the answer more revolves around what are you trying to do with it. If you are a security analyst at a company, you might have a different answer that someone on the network security team, who will have a very different answer from someone who is a malware analyst for an anti-virus company. That being said, there are some commonalities to keep in mind while analyzing malware. These questions should be kept in the back of your mind as many will cross roles and help you plan your next step or steps in the analysis process.

Some of the key general questions are going to be what does the malware do, what damage did it cause, what are indicators of compromise, what is the sophistication of the intruder, what vulnerability or exploit did they use, what network calls does it make, and has anyone seen this before? When we can find malwares purpose and goals, we can help classify the malware and identify the risk and potential attack vectors. When we determine the indicators of compromise we can help establish what it went after, what it could have gotten, and what we can do to reverse or revert any damage caused. This can also help with detecting the malware in your environment and determine who got infected to complete your impact analysis. For example, if the malware always makes a 'tempDB.txt' file and stores it in the AppData folder in Windows; you can start looking for this file to help detect the malware. This can also lead to helping us understand the sophistication of the attacker.

These indicators can be harder to pick up on, but can help you determine if this is malware hitting anyone who is vulnerable, targeted at just your systems, and if it is targeted at your systems is it potentially insider information they are using to gain access? These will help figure out best approaches to take if/when you try to catch the intruder – though often times that part will be left up to your state and federal law enforcement agencies. From analyzing the malware you may find a specific vulnerability. This could be very specific to your systems if it is a sophisticated inside all the way to a published CVE that the script is using to take advantage of a missing patch. As far as looking at network calls it is making, this can help you classify the malware or determine if it is a bot net ‘phoning home’ – this will be discussed later in the paper. The last item on the list is has anyone seen this before. There are various information sharing and threat intelligence vectors that industries will use as well as public methods like twitter and news outlets that will share or link to details of malware. In addition to these, there are websites you can upload samples to that can share more details on if it has been seen before and which AVs potentially already detect it. The pros and cons to these sites will be discussed later in the paper.

If you are working in a corporate environment, questions your business might want to know in addition to those listed above are what data was taken (and what regulations adhere to that lost data), how long has it been here, and how did it enter in the first place? With the first question, one of the key parts are any regulations in place around loss of data or potential loss of data. There are some laws and regulations that have timing requirements such as for notification of breaches. If it was something like a key logger, then there might be additional assessment that needs done outside of the malware to see if those credentials were used. Figuring out how long the malware has been around will help determine scope and exposure to look for indicators of the malware and malicious behavior that could have occurred. Determining how and when malware got into the system will help companies try to find gaps in their current controls and potentially help with training efforts such as if it came in through a phishing campaign.

The last set of questions to keep in mind are more technical in nature and explanations of them will follow later in the paper. Those questions are what network indicators can we find, what host-based indicators are around, is there a persistence mechanism, what is the date of compilation, what is the date of installation, what language is the code written in, what language is it compiled in, is the code packed, is the code obfuscated, is the code designed to thwart analysis, is the code designed to detect virtualization, and does the code have a rootkit?

This is not meant to be an exhaustive list of questions to keep in mind, but rather to help with formation of notes and to ground the researcher with some of the items to be looking at in the malware. This paper will focus on Windows based malware as an introduction, but it should be noted at current state there is malware targeting macOS, tablets and mobile phones, Internet of Things (IoT) devices, and even some focusing on crypto mining by going after large servers or local graphical processors.

CHAPTER 2. STRUCTURE OF MALWARE

As we get into malware analysis it starts to appear that there can be almost an infinite number of structures, types of malware, and setup to them. However, when we use statistics you will find most malware is not that bad to classify. So what are those 125 lines of code made out of? The more common breakdown you will find will have malware made out of 3 components: the replicator, the concealer, and the bomb. The replicator portion of the malware is what it is going to use to spread to other files or other systems. They can "spread via diskettes (and other exchangeable media), shared folders, network scans, peer-to-peer networks or emails and instant messages." (G DATA). The concealer may or may not be present depending on if it is a simple or complex virus. Some of the malware might have built in detection methods to use during replication to see if it is already present on that machine. The purpose of the concealer is to keep the virus from being detected. It can accomplish this in a variety of ways. They might detect if you are running a debugger, if malware is running, trying to be unpredictable by running at pseudorandom times/intervals, add a bunch of superfluous or obfuscated lines of code around their core pieces, spreading themselves out across multiple files, and occupying volatile memory to name a few. The third piece is the bomb. This is the main harmful component of the malware. It is the payload that runs an exploit, will exfiltrate data, cause a denial of service, disrupt the system memory, crash a system, and log your key presses to name a few scenarios. The bomb will let you see the damage area of the malware, and can help you gain insight on the true intent of the malware.

While the malware researcher is looking at those 3 components of the malware, they will be able to attempt to classify it. Malware is often classified based on the nature and functionality of the code. The 3 most common types of malware you will see are viruses, Trojans, and worms. Viruses are usually going to be described by their replicator code. A virus will rely on another program, often attaching itself to or piggy backing off of it. A Trojan will be classified based on the concealer and bomb code. The Trojan will usually conceal itself as a legitimate software, and its bomb logic will usually exfiltrate data. Worms will be classified by their replicator. They can often replicate themselves on their own.

In addition to virus, Trojans, and worms, there are many types of additional malware. Some of the other classifications you might run across are rootkits, backdoors, spyware, adware, ransomeware, downloader, botnets, information stealing, launchare/launchware, scareware, spam-sending, and mining. Rootkits are a type of malware that is designed to gain root access on a machine. They are classified by their bomb code. Backdoor malware is classified by its bomb and partially its concealer. Backdoors are generally intended to try and install a way to bypass authentication, secure a connection, or obtaining access to plaintext. Spyware is malware that will monitor and track a user’s activity, browsing habits, keypresses, and any other data it can potentially use, exploit, or sell about the machine it is on. It is classified by its bomb code. Adware (sometimes also called malvertising) will try and display advertising banners while a program is running that generally might not show ads. Adware is categorized by its bomb code. Ransomware is classified by its bomb code. There are different variations, but ransomware will generally try to prevent access to a system or files (generally via encryption) and hold them ‘hostage’ until a ransom is paid. It should be noted that paying the ransom with this malware doesn’t guarantee that the victim will get the unlock keys or gain access back to their system(s). Downloaders are very similar to Trojans and are mostly defined by their bomb. They will wait until an internet connection is established and try to download more files (often more malware). Botnets are a bit harder to describe than most of the examples. More commonly, they will be built on top of Trojans. Botnets are a group of infected computers (often called ‘bots’ or ‘zombies’) that will call home to a command and control (C&C) center for instructions and control. These can be telnet calls, IRC, P2P, and HTTP calls to name a few. These will be classified by their bomb code. Botnets can be used to distribute malware, email spam, bitcoin mining, spyware, and play a hand in sending out distributed denial of service (DDoS) attacks. Information stealing malware is similar in behavior to spyware but is often times more targeted. It is classified by its bomb, but differs in nature of generally being more targeted. This could be set up to scrape volatile memory to grab payment information, only log keys, proprietary code or information, or screen scrapers to name a few. Launcher/launchare/launchware and loaders all refer to the same type of malware. It “is a type of malware that sets itself or another piece of malware for immediate or future covert execution. The goal of a launcher is to set up things so that the malicious behavior is concealed from a user.” (Sikorski). Scareware relies on social engineering to cause fear or shock to coerce a user to make a payment for a product or to try and blackmail a user. The product they buy may or may not be legitimate, and may or may not be needed. The blackmail threats are generally fake. This can be popups saying ‘your computer is infected by 1500 viruses, click here to remove them’ to software saying ‘we hacked your computer and know you visited *insertname* adult website, pay 1 bitcoin or we will release the video to everyone on your email list’. Scareware is often found via its bomb. Spam sending malware is malware that is classified by its bomb and relies on infecting a machine or server and will try to send out spam or malicious emails.

After having discussed the 3 components that make up most malware and talking over a few classifications, it should be mentioned that the classifications are not mutually exclusive. You will often find some combined or chained together. For example, spyware may have a downloader as well to help download other malware.

In addition to classifying malware by its nature and functionality, we can classify it based off of its target. The two main types are mall malware and targeted malware. Mass malware has been described as the ‘shotgun’ approach to malware – its goal is to affect as many machines as possible. Mass malware is the more common of the two types. It is the easiest to detect and in most cases will be less sophisticated than targeted malware. Targeted malware will usually be designed and tailored to a specific organization, company, or software. Anti-virus programs will often times not detect these as they won’t be as widespread. That being said, they may reuse components of known malware and the anti-virus software may pick up on those signatures. These will generally be more sophisticated and rely on advanced analysis. Because they are more targeted and harder to detect they are generally going to be more of a security threat.

In addition to the above classifications, malware is sometimes also detected and classified via host-based signatures; trying to detect it on the victim’s computer. Network signatures can also sometimes detect malware by monitoring traffic. There are cases of an intrusion detection system (IDS) picking up on malicious traffic.

CHAPTER 3. MALWARE ANALYSIS OVERVIEW

Even though most malware is only 125 lines of code, it is very complex. Even in software development having access to all of the source code it can be daunting to figure out what a program is actually doing. With software development, developers will usually follow a process like the software development life cycle (SDLC). A penetration tester might follow a process or framework like the Attack Kill Chain – more specifically its lateral movement cycle. With malware analysis, a researcher will similarly want to use a systematic approach. The 4 general stages of malware analysis are manual code reversing, interactive behavior analysis, static properties analysis, and automated analysis.

Manual code reversing is related to manually analyzing the code and potentially reverse engineering the code. This will often be viewing the assembly code, trying to decode stored data, reviewing the logic of the program, and helping to understand the capabilities of the malware. A lot of this will also encompass advanced static analysis techniques, and some advanced dynamic analysis techniques.

Static properties analysis will cover basic and advanced static analysis. This process is very similar to static analysis of software development where a developer might scan their source code for bugs, vulnerabilities in 3rd party dependencies, and code quality to name a few. This will be reviewing the malware without actually running it. In another analogy, this can be related to an autopsy of the code – dissecting the 'dead' code. In doing this, the researcher is looking for what the code needs, what resources it is taking advantage of, can we decompile the code, any static PE properties, any system calls, any interesting strings, and any dynamic link libraries (DLLs) that the code is using. More on these later.

Interactive behavior analysis, also called dynamic analysis involves observing the malware running live. In software development this would be similar to using tools like Selenium, OWASP ZAP, or Burp Suite. It has even been related to the ant farms that have glass on both sides you can see the tunnels the ants dig – it involves trying to set up an environment where you can observe the malware. Dynamic analysis will include monitoring network traffic, file system modification, registry analysis, and memory analysis. There is basic dynamic analysis that involves running malware on a system to observe behavior. This does not require deep programming knowledge. However advanced dynamic analysis may involve a bit more programming knowledge and usually revolves around using debuggers to analyze what the malware is doing.

The fourth stage mentioned is automated analysis. Automated analysis involves having an environment set up that can automatically do this analysis. There are commercial tools like VXstream that you can just drag and drop files into or detonate files in and view what interactions they have. There is open source software as well like cuckoo that can be set up to automate it. When we go back to looking at the comparisons mentioned above, 125 lines of malicious code in 70,000 lines of code for a text editor is potentially very hard to spot. Keep in mind that there is a lot of software as well that doesn't have malware in it. This is why what a malware analyst is looking for is often referred to as a 'sample' instead of malware – as it might be clean software.

From this split, we have 4 categories that we can hit. True positive, false positive, true negative, and false negative. In this case, a true positive will be a sample has malware, and the automated tool alerted us that it did. A false positive would be a sample did not have malware, and our automated tool told us it did. A false negative would be it had malware, but the automated tool did not alert us that it did. A true negative would be the sample did not contain malware, and the automated tool did not alert us that it did. With an automated tool, the goal would be to analyze those 4 categories and try and tweak thresholds and scans to try and have the highest amount of true positives and true negatives while trying to keep false positives and false negatives lower. Analyzing one piece of software at a time works in some cases; but for bigger companies that might exchange millions of files a day there are two options with fighting malware: hire tons and tons of malware analysts, or put automation into place and have the malware analysts investigate the true positives that the tool alerts on.

CHAPTER 4. MANUAL LAB ENVIRONMENT

One of the most important decisions to make is what kind of and how an analyst will set up their malware analysis lab, also sometimes called a sandbox. There are two main approaches, physical devices (usually personal computers (PCs), tablets, or actual cell phones), and virtual machines (VMs). The approach that you go will depend on your end goal, budget, and amount of space that you have.

With a physical lab, you would need a device running whichever operating system or environment you were wanting to test. This could be a car with the OS and version running on it, each cell phone needed (iPhone for each version you want to test, various android versions), or a set of multiple personal computers. This can cost a lot, and take up a lot of space. In addition to this, the researcher will want to configure a network that is isolated for the devices to use depending on which category of malware they are studying.

Since this paper focuses on Windows, then using the above the lab the researcher would set up would involve one or more personal computers. Getting each machine to have the same operating system, all the patches to the point we need, and then get baselines of them can be time consuming. There are tools that can be used like Truman to help automate re-imaging of machines. Another option is to get hard disk drive (HDD) write cache cards to help with this.

The pros of this type of lab environment is that it is more realistic – the researcher will get to see what the sample does on the actual hardware in the actual environment. The cons are it can cost more, take up more space, and take up more time.

The alternative to this that is increasing in popularity is the use of virtual machines. With virtual machines there are a lot of free options that can be used. There are lots of operating systems that have virtual machine editions that can be simulated. Some of the potential programs are VMWare, Parallels, Xen, and Microsoft Virtual PC. A few of these such as VMWare support the idea of taking a snapshot.

A VMWare snapshot will basically take an image of the time the computer is at. When the researcher restores to this point, everything that was done after it will be gone – all new files created, all registry changes, all text files, all system file changes, etc. It will be as if none of it happened. It should be pointed out this is very different from a Windows System restore. A system restore will generally just restore system files to a previous state. This means if there was a Microsoft Word document with malware – it would generally not be touched by a system restore.

For added protection, while a researcher is doing static analysis, they can use a different operating system that the one they believe the malware is targeting to start their analysis. This can help prevent an accidental double click of the file.

Though virtual machines have many pros such as only needing one personal computer or server to run on (assuming it has enough resources), being cheaper, and faster to load/restore, there are some drawbacks to using a virtual machine. One of the first is it can be nontrivial to set up and configure the network side of them. Doing one virtual machine at a time it is usually recommended to configure it for a host only network. Depending on the needs, a researcher can also configure them for virtual networks. The second drawback to keep in mind is that virtual machines are not bullet proof. There are simple built in commands a user can run such as ‘wmic bios get serial number’ that will display a serial number on a normal version of Windows, but will display 0 on most virtual machines. There are open source tools built that a researcher can run to determine how easy/hard it is to detect that a virtual machine is running, and give clues and hints on what can be done to make it harder to determine it is in fact running in a virtual machine. That being said, it is a double edged sword. An attacker that is sophisticated that is not wanting to target malware can potentially use these same tools to try and help their malware get around them. Some malware will detect it is running in a virtual machine and do nothing, or act differently. That being said, there are some companies that each user logs into a virtual machine to work as it is easier to license and monitor software in a virtual machine for the company than to monitor physical machines – so the malware may not care if it is in a virtual machine. Virtual machines can have flaws and bugs in their software as well which can help or hurt in analysis. One of the other big flaws with this is that the virtual machine software itself such as VMWare can potentially have 0 day exploits in them. In addition to those and other flaws, it is possible for malware to jump outside of its virtual machine and infect the host machine. A 0-day worm that can exploit listening service on a host operation system will escape the sandbox.

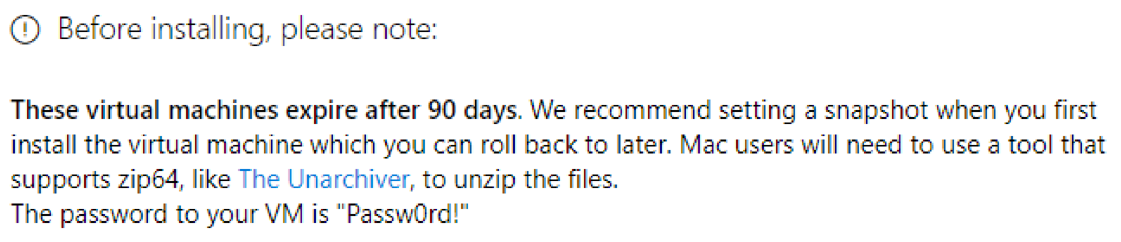
In addition to setting up the environment, it was briefly mentioned about setting up a networking environment for it. This is an area of debate and what state the researcher is in should help decide this information. Connections can be opened up for certain network traffic (such as hypertext transfer protocol (HTTP) to help install certain tools, ports to allow windows and software updates, and file transfer protocol (FTP) ports to get the sample file on the machine). However, when performing the analysis, the network should generally be closed off. Often it is referred to as ‘calling home’ or ‘phoning home’ if the sample be analyzed is in a certain category. Letting the malware call home can make figuring out what it is doing a lot easier as it will behave normally. That being said, it might give the attacker the address of the machine it is running on (which can make that computer the target of additional attacks). This can potentially accidently enter you into a real time battle if the machine pops an exploit and connects to a control server. Because of this risks, it is generally advised to keep the network isolated and create ‘in house’ services to respond to the calls home. If a researcher knows what they are doing, a potentially safer route in letting it actually call home may be to connect the gateway of the service to an anonymized network such as the onion relay (also known as the onion router and TOR).

At the time of writing this paper, students at Iowa State should have free access to the professional version of VMWare which will allow them to set up a lab at home, and in the future potentially be able to take these same virtual machines and set them up on ISEAGE for students to do basic malware analysis.

Setting up the Lab

The current place to start would be to make sure the host operating system is fully up to date. Check and double-check everything is patched with the latest firmware and software. The current address for Students is https://cytools.iastate.edu/vmap/. This should prompt for a SAML login and then connect to a onthehub.com website where students can get various software. At the time of writing, the latest version of VMWare Workstation Professional edition on there is 15.

Once that software is installed, the next step is to get a Windows operating system image to install in VMWare. Again, there are ways to emulate iOS and Android – but these are out of scope for this introduction to malware analysis. That being said, with the amount of phones and tablets coming out and gaining popularity, it is a growing area and gaining lots of market share; especially with the internet of things devices as well. Keep in mind what attackers are wanting to go after and their various methods. If they are going for easy money and doing the mass malware or shotgun approach; they will look at what has the highest market share and go after that. For Microsoft, you can Google it; but https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/ is the current location to download a free VM to test malware with. You can select operating systems based on which browser you would like. This is allowed and legal. It should be noted some sites that


Before installing, please note:

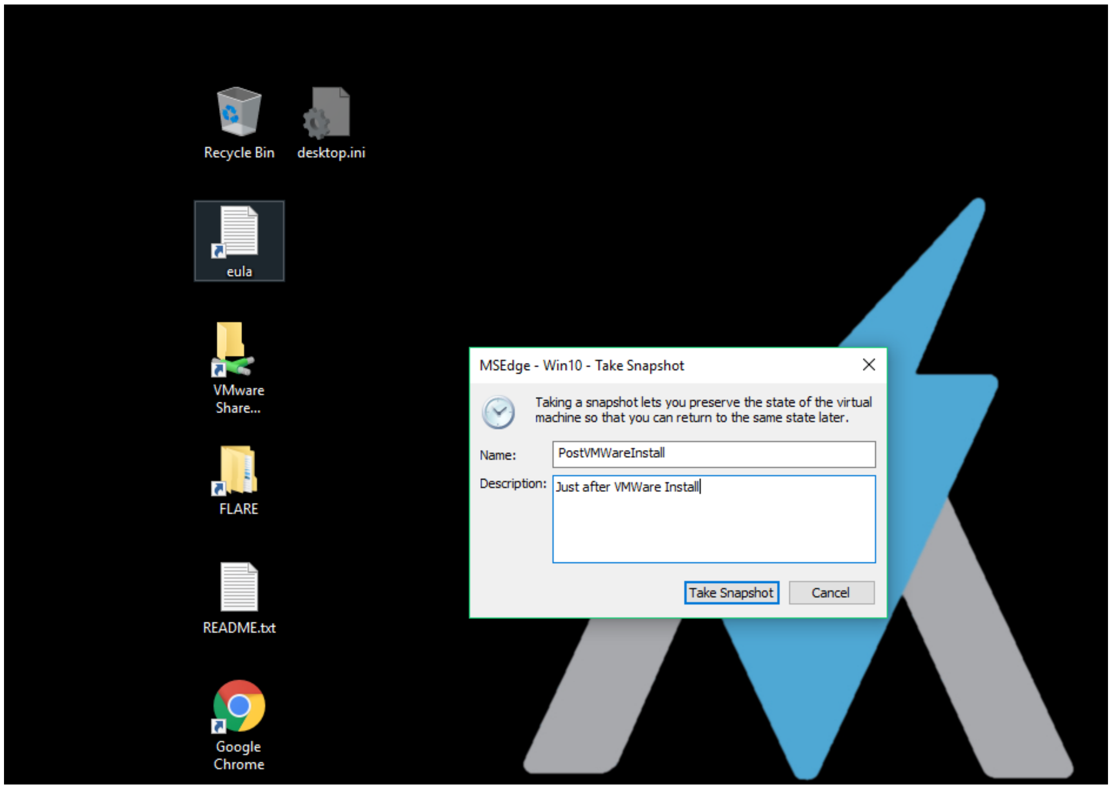
These virtual machines expire after 90 days. We recommend setting a snapshot when you first install the virtual machine which you can roll back to later. Mac users will need to use a tool that supports zip64, like The Unarchiver, to unzip the files.
The password to your VM is "Passw0rd!"

Figure 4.1 *Important note on Microsoft's website on password for the virtual machine and when the image expires.*

a person visits to learn more about malware analysis will recommend torrenting a version of Windows that is cracked and has a 'valid' product key. A majority of those methods are illegal – and potentially contain malware. The only big operating system the above site is missing in the Windows suite is Windows XP. The above site will only have supported versions of Windows and XP was 'end of life'ed (no longer supported) April 8th, 2014. Malware like WannaCry and Petya are relatively recent and did in fact target Windows XP. Note that in figure 1 the VMs expire after 90 days, and what it says their password is for initial use. Over time the password will change, so do not rely on figure 1. As of my latest check September 1st, 2018 on https://www.statista.com/statistics/218089/global-market-share-of-windows-7/ Windows made up 82.45% of desktop OS market share and MacOSX was 12.89%; so this should cover over 95% of users and most of our students.

The author has not used nor tested it, but there is VMWare Horizon for Chromebooks that is supposed to be very similar to VMWare Workstation to help cover some of that remaining 5% of users. At last check, the Windows OS download was 4.5 gigabytes (GB) in size; so this can take some time to download depending on your connection speeds.

The book Practical Malware Analysis lists a ton of great tools on page 465. Finding the tools individually yourself can actually take quite some time to locate and install. However, the company FireEye created flare-vm, a package that contains a lot of common malware analysis tools; most of which are listed in the book; and a few that are not. This is by far the easiest add on to use to gather all the tools to use for manual analysis. If a



Taking a snapshot lets you preserve the state of the virtual machine so that you can return to the same state later.

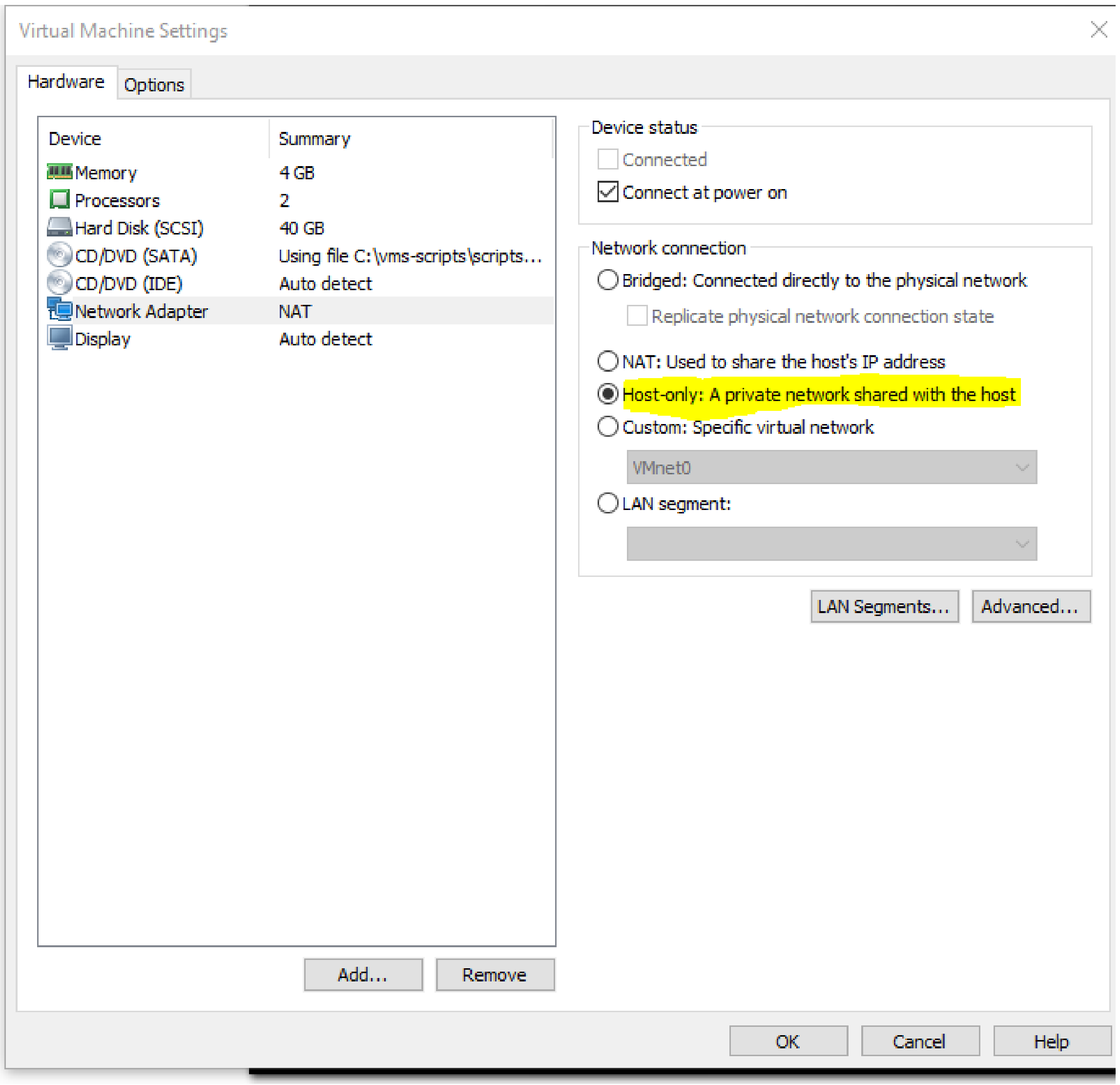
Name: PostVMWareInstall

Description: Just after VMWare Install

Figure 4.2 *Screenshot of the flare-vm background and taking a snapshot in VMWare Workstation Pro*

researcher is wanting to mimic their production environment, flare-vm might not be the best approach; but for the purposes of beginning malware analysis, this makes a great tool.

Once you have downloaded the Windows virtual machine and installed it, boot it up and login. Open Internet Explorer (note if you are using Windows 10 you want to make sure



Virtual Machine Settings

Hardware | Options

Device	Summary
Memory	4 GB
Processors	2
Hard Disk (SCSI)	40 GB
CD/DVD (SATA)	Using file C:\vms-scripts\scripts...
CD/DVD (IDE)	Auto detect
Network Adapter	NAT
Display	Auto detect

Device status
- [ ] Connected
- [✓] Connect at power on

Network connection
- ○ Bridged: Connected directly to the physical network
  - [ ] Replicate physical network connection state
- ○ NAT: Used to share the host's IP address
- ● Host-only: A private network shared with the host
- ○ Custom: Specific virtual network
  - VMnet0
- ○ LAN segment:

Figure 4.3 *Screenshot of VMWare Workstation Virtual Machine Settings, highlighting setting the network to Host-only.*

you are using Internet Explorer and not Edge), and go to http://boxstarter.org/package/url?https://raw.githubusercontent.com/fireeye/flare-vm/master/flarevm_malware.ps1. Select Internet Explorer if it prompts you on how/what to open the file with. When the download is completed, click run. Note what the default

Windows background looks like. We will know flare-vm install is done once that changes. The author stopped watching during this process, but the virtual machine will restart multiple times as it installs new tools, and took a little over 4 hours to install. You will know the process is completed when it says ‘Type ENTER to exit’ and the background is a flare-vm one, as seen in figure 2. At this point, you will want to open the virtual machine settings and click on ‘Network Adapter’ and set it to Host only as seen in figure 3. The next step at this point is to take a snapshot. You will want to go to the menu at the top, click VM, then Snapshot, then new snapshot. You will want to name this appropriately, such as ‘PostVMWareInstall’ or ‘Baseline’. At this point, you are done setting up your virtual machine and have taken a snapshot. You can take more snapshots as time goes on if you ever want to go back to them such as ‘postSampleRun1’.

CHAPTER 5. STATIC ANALYSIS

Again in quick summary, static analysis is where we will be figuring out anything we can about the sample before actually running it. Most malware samples will be provided to you in a compressed format, and renamed. The common password to open a file is “infected” without quotes. Most will use 7zip to compress them. A lot of times, the samples that are found are in a lab environment with their extension renamed, such as renaming an executable files to .exe1 instead of its standard .exe. This is to prevent accidental double clicking and running the sample. In a production environment that may exist at a company, this may not be the case so the researcher would want to exercise extra caution. A researcher can search the internet for collections of samples to view. Upon searching, some potential candidates on GitHub are:

- https://github.com/mikesiko/PracticalMalwareAnalysis-Labs
- https://github.com/fabrimagic72/malware-samples
- https://github.com/ytisf/theZoo
- https://github.com/InQuest/malware-samples
- https://github.com/rshipp/awesome-malware-analysis

Since the author used Practical Malware Analysis as a main source, most of my examples will use samples from their collection. Static Analysis is a great starting point as it can help a researcher learn a bit more information about the sample that they can use to augment, add to, or tweak their environment with.

Basic Static Analysis

For basic static analysis, we will go over anti-virus (AV) scanning, hashing, strings, packed/obfuscated code, portable executable (PE) files and headers, and libraries.

For anti-virus scanning, one of the first steps a researcher can do is to scan the file with an anti-virus program and see if there are any results. If it is a malware sample from one of the locations listed above, chances are very high that it is in quite a few anti-virus programs; Windows might have even detected it as the file was transferred over and unzipped with its built in program. For a new researcher, this is a great starting point as the anti-virus program might give hints to what category the sample contains, such as Trojan or key logger. Another popular tool is to upload the sample to VirusTotal. VirusTotal will scan files and uniform resource locators (URLs) with, at the time of this writing, over 70 anti-virus scanners, then display the results. This is a great resource to use, and sometimes users will even leave comments on samples that can be helpful. However, VirusTotal has two big cons to be aware of. The first is that VirusTotal is public – if you upload a sample that hasn’t been seen before, the writer of the malware may be able to see this and see you are investigating their malware. The second, is that paying customers can download the uploaded samples to do research with. This means if you have a lot of personably identifiable information - such as an excel file with customer first names, last names, social security numbers, addresses, and date of births – and upload this to VirusTotal as you think it contains malware; a paid customer can potentially download that file. This will generally be considered a loss of customer data for the researcher and potentially cause extra work. Because of this, always think twice about what you are uploading and if it should be uploaded so that data is not made public that shouldn’t be.

Often times, you can get MD5 checksums or hashes of the file and upload those to sites to check as well, or sub portions of the file. The other side to hashing will be to set a baseline of pieces of information while you do dynamic analysis. However, in this context we are talking about using hashing to fingerprint the samples and other files. By doing this, it gives common ground to verify other researchers are talking about the same file/sample. Also, if you have malware that manipulates a text file to add data to it for example, taking a hash of the file at the current time, then hashing it after running the malware you will see a difference in the hashes. Thus the hashes can be used to quickly identify differences. A free tool to do this is md5deep. A researcher would just run the command ‘md5deep C:\path\to\file.exe’ without quotes. There are graphical user interface (GUI) versions as well.

Strings in malware are what you would expect a string to be in programming. They are a sequence of characters. “A program contains strings if it prints a message, connects to a URL, or copies a file to a specific location.” (Sikorski). A researcher can run strings on a file by using the command ‘strings filename.exe’ (and you can use ‘> output.txt’ on the end to output the results to a text file; helpful with a lot of the command line programs) without quotes. Strings can be hit or miss on it. It may have a lot of strings that mean nothing such as ‘M{C’, but can have very helpful information. Sometimes the output can be calls from a Windows library such as ‘GetLayout’ or ‘SetLayout.’ They can also be dynamic link library (DLL) names, such as ‘GDI32.DLL.’ An analyst can use some of these DLLs to determine what the program might be doing. One example is if a program uses Wininet.dll, this library is sometimes used to do FTP, HTTP, or NTP. If this DLL shows up in a program that shouldn’t be connecting to the network; it can be a sign that it may be trying to exfiltrate data or receive a file. Another item a researcher might find in strings are hardcoded internet protocol (IP) addresses. Finding one or more of these can help a researcher set up a lab environment where it responds to the malware when it tries to send information. One final item that may be found in strings are error messages. The example Sikorski uses is if there is a string stating there is a missing mail system DLL, this can mean the sample is trying to use a mail DLL to send emails. This could be in a key logger for example, to send the information out once it is collected.

If strings doesn't seem to yield any results, this is a potential indicator that the sample is packed or obfuscated. Packed code will usually have a wrapping program that can unpack the code. This may still use some Windows DLLs to do this. This is one way malware can try to hide from anti-virus programs, as strings are not visible in the packed portion of the executable. PEiD is a common tool that can be used to view what program was used to pack the executable, when it was packed, and other information about the program. If this tool or a similar one works, a researcher can attempt to download that same packing program and unpack the sample. Once it is unpacked, they can go back a step and retry running it through an AV scanner, then hashing it, then running strings on it.

Portable Executable (PE) is a data structure format that executables (.exe) use in Windows and contain the runnable code and a list of DLLs. There are 3 ways to link the DLLs; static, runtime, and dynamic. Static linking imports the code itself from the DLL. This is not common in Windows, but is with some UNIX and Linux programs. One downside to this is that the used code library is copied in, making it hard to differentiate between the original code and imported libraries. Another is it can increase the file size of the executable. With dynamic linking, the code will load all the linked libraries when the file first loads. Think of this like doing role call before starting a project – the manager is making sure everyone is there before they start work. This is most common, as the program stays a smaller size, and can determine it is missing libraries before trying to run and crashing. Runtime linking is more common in malware, and it will not attempt to load a library until it calls it in the code. One program to explore dynamic linked libraries is Dependency Walker. This program can be downloaded, but is also sometimes included with some development kits in windows, such as Microsoft Visual Studio. Dependency Walker will allow a researcher to view the executable, each dynamic linked library, and each function call within the DLL that is referenced. In addition, it will often list versions of the linked DLL and can show timestamps of, checksums, entry points, assembly architecture, and other helpful information. One helpful callout in Sikorski's book is naming conventions in Windows. Be sure to search around online when researching them as there are some things that can be confusing. One example is a function name listed in Dependency Walker might have a W or A at the end, but doesn't in their online documentation. This just indicates that the one ending in an 'A' takes ASCII arguments and the one ending in a 'W' takes a string. Also, they can end with an 'Ex' to indicate it is not backwards compatible, and this is the new call. An example would be CreateWindow versus CreateWindowEx. Another tool that can be helpful is PEview. With PEView, a researcher can view more information about the headers including some of the .text, .data, .rdata, and .rsrc sections. The big take away with this tool is often times if you can view the virtual size and size of raw data they should be roughly close. If they are not, that is usually an indicator that something is amiss.

Using the static analysis techniques described above, a researcher can start to understand functions the code may use, potential IP addresses it will reach out to, potentially when it was created, and some basic header information. At this point, they could use other tools like Yara to write rules to detect the sample based on some of the information they have gathered.

Advanced Static Analysis

Advances static analysis is partially out of scope for this paper as it covers a lot of topics and can go quite in depth. At a brief high level, some of the items in advanced static analysis are disassembling the code, Windows registry, Windows API, code constructs, networking APIs, following malware, and using IDA Pro.

There is an entire course at Iowa State on reverse engineering that the author has not been lucky enough to take. From the description it is believed the course would hit on disassembling the code, some debugging, and IDA Pro usage. From that, learning assembly from scratch can seem like a daunting task and can be quite different than high level programming languages that might be more common to software developers. However, knowing a lot of the code constructs can help follow and understand assembly. Examples are for loops, while loops, if statements, nested if statements, function call statements, and jumps. Interactive Disassembler (IDA) Pro is a unique tool that can be used for debugging, static property analysis, and decompiling code. There are also plugins to help it support many languages. It even has a graphical mode a user can go into where it maps out the program flow and calls. A similar tool to this mapping would be Binary Ninja. The flow and mapping can help a researcher figure out what is going on, especially if they have a bit weaker assembly programming skillset.

Windows registry stores a lot of variables, information, and settings for programs, hardware devices, user preferences, and the operating system to name a few. Some malware may target going after the registry for various reasons. A main reason may be to set the malware to run automatically. The registry could also be tweaked to point to different executable files when performing certain functions. From this, when looking through strings, disassembled code, or debugging code, look for calls like ‘HKLM’, ‘RegOpenKeyEx’, or ‘RegSetValueEx.’ These changes can be damaging and even removing malware often time’s registry settings are not reversed which can cause crashes farther down the road.

Windows has an API within it that can allow a program do tons of different functions. For better or worse, the API is so huge that there is little need for 3rd party extensions of it. This means programs interacting with it will use a defined set of names. The Windows API can allow a program to access the Windows Registry mentioned above, create files, read files, write to files, display values to a window, access network functionality, and access shared files to name a few functions. Becoming familiar with some of the API calls can help find relevant strings and help figure out what calls are being made when disassembling code.

The networking API mentioned above is how most Windows based malware will connect to the network. The networking API allows programs to open sockets, open connections, send data, receive data, accept connections, and listen on a socket to name a few. Finding these calls can allow a researcher a better chance at narrowing in on malicious traffic and potentially spoofing responses to see how the sample will react.

Though we have touched on a lot of advanced static analysis at a high level, it should be noted that a researcher will still need to keep in mind what they are researching. For example, researching the calculator on Windows (calc.exe), it may not need to connect to the internet at all; so finding network calls could be a sign of compromise. However, it may have help documents that do connect to the internet to download the latest version of their text, or to link to an online knowledge base. When looking at Windows native notepad versus Notepad++, Notepad++ will have a lot more connection based code for it to check for updates, have a plugin manager enabled, and connect to a plugin store. It will also need to download plugins for itself. This means that in manually researching a sample, keep in mind what all the program can and is capable of doing to not take a path that could lead nowhere. There will be times where finding this information leads to finding out more about those 125 lines of code in the malware sample, and many more that lead to investigating functionality of a program that was intended to be there and used in that way.

CHAPTER 6. DYNAMIC ANALYSIS

Dynamic analysis is performing analysis during or after a sample has been run. This is part of the interactive behavior analysis. Dynamic analysis will rely on some pieces that are set up and learned in the static analysis portion. It can be broken into basic dynamic analysis and advanced dynamic analysis. Advanced dynamic analysis will rely mostly on debugging and bleed over into some of the manual code reversing category.

Basic Dynamic Analysis

Basic dynamic analysis will consist of monitoring processes, will rely on a sandbox environment (which is the virtual machine environment set up in chapter 4), can use an external sandbox environment, compare registry snapshots, faking a network, and packet sniffing.

For process monitoring in Windows, one of the go to tools would be ProcMon. It was built to combine two tools that are no longer supported, RegMon and FileMon. To use ProcMon a researcher would simply launch the program, then click File then Capture Events and it will start. They would then double click on the sample or use command prompt to run it. Generally the researcher would want to wait some time before stopping the capture as some malware will not react right away. The output could be exported if desired. The output can also be filtered to better see what is going on. Though ProcMon can be used to monitor some network activity as well, it is not recommended to use the tool for that. There are inconsistencies between versions of Windows, and other tools that are better suited for that. Procmon will show sequence, time, process name, pat, operation, and result which can be helpful in tracking down what ran. Using its filter you can study registry changes to see how it installs itself and file system changes to see what files it modified, created, or deleted. In addition, you can view process activity to see if the spawned different or additional processes. Lastly, filtering on networking may allow the researcher to see what calls were made or open ports. Another tool to use is Process Explorer. This is a Microsoft tool. One key feature is to watch this and see what process IDs (PIDs) show up that are new. Another useful feature of this tool is to verify if a piece is signed by Microsoft. This will check the on disc file though and not what is loaded and ran in volatile memory.

Sandbox wise, a researcher can use their VMWare environment to run, rerun, and take snapshots of before and after a sample was run. In addition to those, there are websites that have sandboxes set up, and commercial tools that spin up sandboxes to detonate samples in. There are a few drawbacks to these. The first is that a researcher has to be willing to upload these samples to a website if they are going after free services (such as Norman SandBox or GFI Sandbox). This runs into the same problem with VirusTotal where if there is company confidential information in the sample, it might not be the best idea to upload it. Another drawback is they will simply run the executable or open the file – so command line arguments cannot be specified. They do, however, provide an organized report that can be useful and used for initially getting a bearing on what the sample does. The output report may include things like behavior traits, mutexes, registry activity, network activity, and virus total results. Another drawback is some sites might not let you pick the operating system the sample runs in – so if the sample only works in Windows 7 but it runs it in Windows 10 a researcher might not get accurate results. The sandbox also might run too fast to allow it to properly detect the sample to unpack itself, download and install its required components, then run.

Registry snapshots are another helpful piece in analyzing a sample. Using a tool like Regshot, a researcher can simply open the tool and hit the first shot button. This will capture the registry and get any relevant information. The researcher would then run the sample, and click the second shot button, then the compare button. This comparison can be exported if needed. Generally, these snapshots will be used in addition to a tool like ProcMon to piece together what changes are being made.

Faking a network can be very beneficial in figuring out what calls a sample is making and what information it is trying to gleam. The tools mentions next can be configured and run on the same local virtual machine instance that a researcher is testing a sample on. Some malware may be built to detect this, so a potential safer approach is to set up this network on a different machine and configure your network accordingly. The first tool to use is ApateDNS. This tool will listen on UDP 53 on local machine by default and if it is configured will spoof domain name system (DNS) responses. Running locally the researcher will have easy instant access to the GUI to view requests. Another tool that can be used and setup is INetSim. INetSim is Linux based, but will simulate a lot of common internet services. These can include DNS, HTTP, FTP, IRC, ECHO, and SMTP. Another tool to use is netcat. If a researcher can figure out what calls the sample is making, they can use a slew of simple netcat commands to set up listening or setting up inbound and outbound connections. However, be cognizant of the machine you are running netcat on. If you are analyzing malware such as a worm and allow it to connect to a machine not in your safe network via netcat you can accidently spread the malware.

The final piece of basic dynamic analysis is sniffing network traffic with a network sniffer. Though there are a few, the most common go to will be Wireshark. This can be done by running Wireshark locally on the virtual machine, or setting it up on a different machine running one of the other network simulated services. It should be noted that if it is running on the same machine the sample it on, the sample may detect this and not work, or try to run a Wireshark exploit.

It is worth calling out in Sikorski's book they recommend when doing basic dynamic analysis to run ProcMon, set a filter on the sample executable, start process explorer, use Regshot's first shot button, set up the virtual network and sniffing/logging, then run the sample. After this, the researcher can finish up the capture process with each tool and export any reports needed and start going through results. This can help save time when analyzing the sample.

Advanced Dynamic Analysis

Advanced dynamic analysis generally consists of debugging the sample. It is partially out of scope for beginners with malware analysis, but it will be reviewed at a high level. There are source level and assembly level debuggers. Most of the debugging done for malware analysis will be assembly level debugging. It will require decent knowledge of assembly. Though debugging sounds simple, it is no small feat to master. There is stepping over code versus stepping into code. Setting breakpoints, exceptions, modifying execution, using the right debugger, and being able to understand what the code is doing. One common tool that may be used if it is x86 assembly code is OllyDbg. OllyDbg has a lot of features it is capable of, some of which being a 4 panel interface, memory map, conditional breakpoints, loadings DLLs, tracing, exception handling, patching, and plugins. Another popular tool that would be used for this is WinDbg. This is a free Microsoft made debugger. Sikorski's book states that OllyDbg us more popular for most assembly debugging, but that WinDbg has the advantage when it comes to kernel debugging and rootkit analysis.

CHAPTER 7. AUTOMATED ANALYSIS

Though some sandboxes were mentioned in the dynamic analysis section, it should be noted that there are some sandboxes that can be purchased for companies to use on premise. One popular open source one however is Cuckoo Sandbox. The details of its installation will not be gone over as it is slightly more than 40 steps depending on which path is taken (https://cuckoo.readthedocs.io/en/latest/installation/). It can be hosted on Windows, Mac, or Linux. It does have a few dependencies required on the host machine. Once those are installed, it then needs a guest machine VM setup with a guest OS and needs to be configured with a username and password. Once everything is configured, there is a local web interface that can be used to upload a file to. Cuckoo will then take the file, spin up the guest VM, detonate the file, and output the report. This is something that could be set up and integrated in with ISEAGE as well if desired – but the free capabilities seem to cover potential needs for now. A piece that is worth calling out about these tools however, is that some companies have started integrating APIs on top of these services to try and do real time analysis on files. Some have even used these to integrate their honey pots to send samples over to collect and gather current and new samples.

CHAPTER 8. COMMERCIAL TOOLS

As mentioned before, there are many commercial tools that can be purchased to help automate the malware analysis process. Many can help speed it up, but there are still pros and cons to it. A researcher after learning even the basic steps to malware analysis can look at output of some of the tools and have a sense on if the tool is giving false positives, or false negatives. It can also help a researcher appreciate the amount of time saved by not having to do it manually.

The second piece to take into consideration is cost of the tools. As useful and helpful as some of these tools are, a company with 4 employees making $10,000.00 a year in profit would not be able to afford an entire suite of commercial grade malware analysis tools. For companies like those, a starting point might be to rely on free tools. Open source tools might not be the most up to date and easy to support at times, but they have a lower entry cost and can help a growing company grow into better tools and support. This will help them balance their money invested into security with their risk appetite and their confidentiality, integrity, and availability needs.

CHAPTER 9. FUTURE WORK

With limited time, there is a ton more work on the topic that would have been fun to cover and go into more depth on. The goal was to get a common starting place future students can use to start analyzing malware samples and apply some of the knowledge learned from other courses taken. The main end goal would be to integrate this within ISEAGE or flesh out the work enough to have a semester long course on malware analysis, assuming there is enough interest. Without taking the 538 reverse engineering class, the author is unsure how much crossover there would be between that and a lot of the advanced static analysis techniques. This is definitely an area that could expand to tens of pages. In addition, screenshots could be added of each tool mentioned that contains a GUI, but that could clutter up the paper. Videos were attempted to be captured of the tools with audio talking about what was going on, but the author could not get the software to work as intended – it either froze on a screen and recorded only audio, or recorded the video with no audio. Advanced dynamic analysis could expand quite a bit with all of the debugging it goes over as well. In addition to those two main points, there could be a lot more information added around malware behavior and have specific examples of each added. A section could go over covert malware launching techniques so a researcher knows what to look for. There are tons of encryption and encoding techniques such as Caesar Cipher, XOR, and base64 that can be used to encode payloads as well as custom encoding. There was tons of interesting information on using an intrusion detection system (IDS) such as Snort to try and detect some malware information based on network signatures. Sikorski’s book also goes over indirection tactics, operations security, and safely investigating an attacker. More time could be spent going over packers and unpacking code as that is very important in static analysis. Also, anti-disassembly, anti-debugging, and anti-virtual machine techniques would be great to cover more in depth as most new samples a researcher comes across might not be as straight forward as some of the ones provided in the labs and download sites listed above. Shellcode analysis would be another topic to go more in depth on for researchers to get a better understanding of what they might be looking for and how to find it. In addition to that, I think the section on commercial tools and automated malware analysis could be expanded upon to highlight some of the pros and cons. Being a tool, there are still some pitfalls to automated analysis and tying back to the beginning, there will always be the battle of false positives, false negatives, true positives, and true negatives. If a researcher knows how to investigate malware manually, it can make it easier to spot anomalies or go further when automated tooling has gaps.

REFERENCES

Eagle, C. (2011). *The IDA Pro book: The unofficial guide to the worlds most popular disassembler*. San Francisco, CA: No Starch Press.

G DATA Software AG. (2016, September 12). Malware categories. Retrieved from https://www.gdata-software.com/seccuritylabs/information/malware-categories

Isaac, D. S., & Isaac, M. J. (n.d.). The SSCP Prep Guide. Retrieved from https://books.google.com/books?id=V2Skk_UyuKAC&pg=PA341&lpg=PA341&dq=malware concealer bomb replicator#v=onepage&q=malware concealer bomb replicator&f=false

Kendall, K. (n.d.). Practical Malware Analysis. Retrieved September 13, 2018, from https://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Paper/bh-dc-07-Kendall_McMillan-WP.pdf

Kendall, K. (n.d.). Practical Malware Analysis. Retrieved September 13, 2018, from https://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Presentation/bh-dc-07-Kendall_McMillan.pdf

Ligh, M., Adair, S., Hartstein, B., & Richard, M. (n.d.). *Malware Analysts Cookbook and DVD : Tools and Techniques for Fighting Malicious Code*. Wiley.

Ligh, M. H., Case, A., Levy, J., & Walters, A. (2014). *The art of memory forensics: Detecting malware and threats in Windows, Linux, and Mac Memory*. Indianapolis, IN: Wiley.

Njenga, M. (n.d.). Fundamentals of Malware Analysis. Retrieved from https://www.oreilly.com/library/view/fundamentals-of-malware/9781788390279/

Poston, R. (2010, December 19). How large is a piece of Malware? Retrieved from https://nakedsecurity.sophos.com/2010/07/27/large-piece-malware/

Sikorski, M., & Honig, A. (2012). *Practical malware analysis: The hands-on guide to dissecting* *malicious software*. San Francisco (California, EEUU): No Starch Press.