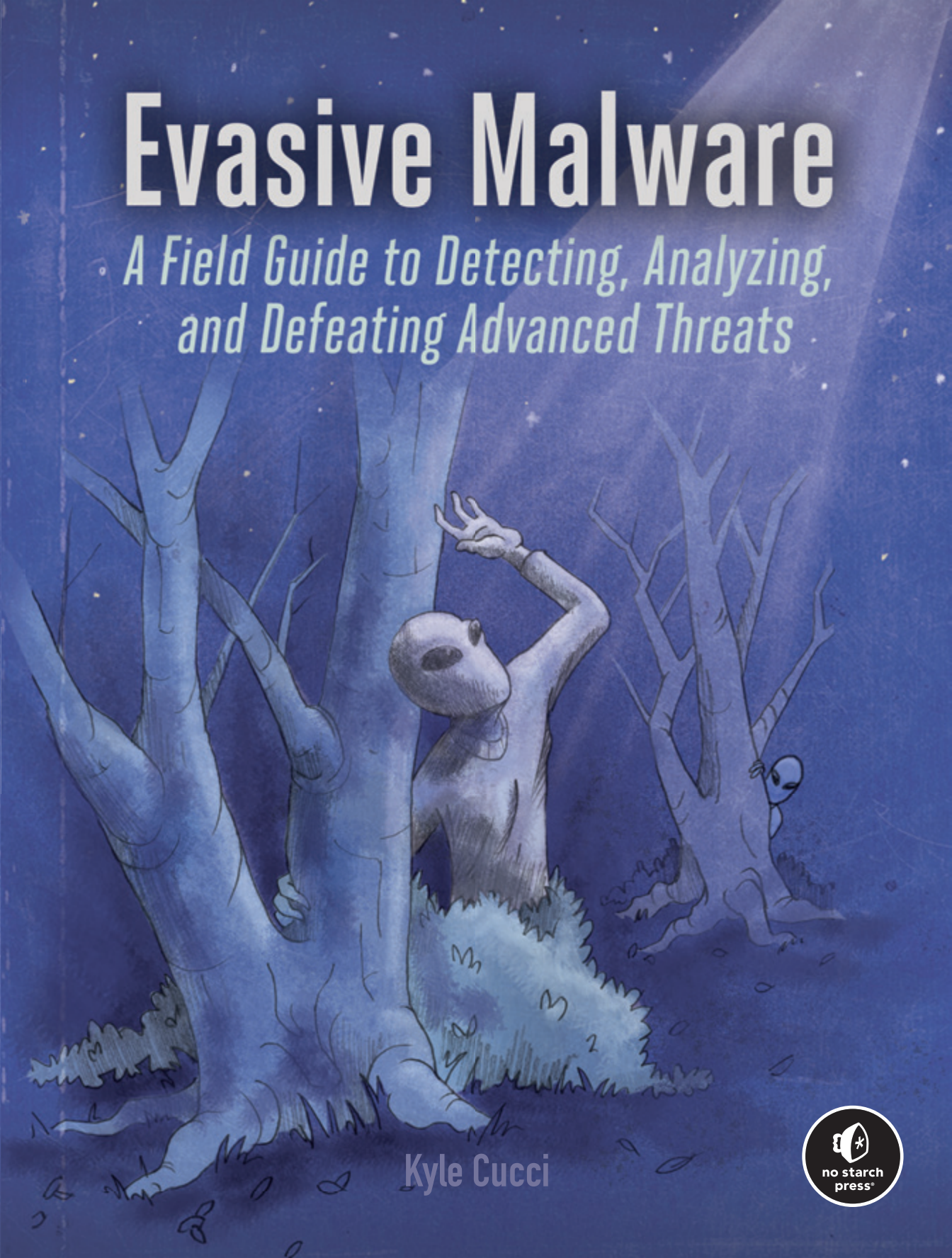
Evasive Malware

A Field Guide to Detecting, Analyzing, and Defeating Advanced Threats



Kyle Cucci

no starch press®

EVASIVE MALWARE

A Field Guide to Detecting, Analyzing, and Defeating Advanced Threats

by Kyle Cucci

no starch press®

San Francisco

ISBN-13: 978-1-7185-0326-7 (print)
ISBN-13: 978-1-7185-0327-4 (ebook)

Published by No Starch Press®, Inc.
245 8th Street, San Francisco, CA 94103
phone: +1.415.863.9900
www.nostarch.com; info@nostarch.com

Publisher: William Pollock
Managing Editor: Jill Franklin
Production Manager: Sabrina Plomitallo-González
Production Editor: Sydney Cromwell
Developmental Editor: Rachel Monaghan
Cover Illustrator: Rick Reese
Interior Design: Octopod Studios
Technical Reviewer: Thomas Roccia
Copyeditor: Doug McNair
Proofreader: Audrey Doyle

Library of Congress Cataloging-in-Publication Data

```
Names: Cucci, Kyle, author.
Title: Evasive malware : understanding deceptive and self-defending threats
   / Kyle Cucci.
Identifiers: LCCN 2024001989 (print) | LCCN 2024001990 (ebook) | ISBN
   9781718503267 (paperback) | ISBN 9781718503274 (ebook)
Subjects: LCSH: Malware (Computer software) | Computer security.
Classification: LCC QA76.76.C68 C83 2024  (print) | LCC QA76.76.C68
   (ebook) | DDC 005.8/8--dc23/eng/20240830
LC record available at https://lccn.loc.gov/2024001989
LC ebook record available at https://lccn.loc.gov/2024001990
```

For customer service inquiries, please contact info@nostarch.com. For information on distribution, bulk sales, corporate sales, or translations: sales@nostarch.com. For permission to translate this work: rights@nostarch.com. To report counterfeit copies or piracy: counterfeit@nostarch.com. The authorized representative in the EU for product safety and compliance is EU Compliance Partner, Pärnu mnt. 139b -14, 11317 Tallinn, Estonia, hello@eucompliancepartner.com, +3375690241.

About the Author

Kyle Cucci has been hooked on computers since building a PC and buying a C++ book as a teenager. He has over 17 years of diverse experience in cybersecurity and IT, and he is currently part of Proofpoint's Threat Research team, with a day-to-day focus on hunting and reverse-engineering malware. Previously, Kyle led the malware research and forensic investigations team at a large global financial institution. Throughout his career, Kyle's threat intelligence contributions and research have been featured in government intelligence reports and security tools and products. Kyle regularly speaks at security conferences and has led international trainings and workshops on topics such as malware analysis and security engineering. In his free time, Kyle enjoys contributing to the community via open source tooling and blogging, spending quiet time with his family, and brewing acceptably drinkable beer.

About the Technical Reviewer

Thomas Roccia (aka @fr0gger_) is a seasoned threat researcher who has worked on complex malware cases and investigations around the world, ranging from cybercrime to nation-state level outbreaks. Thomas has worked with top tech companies, including McAfee and Microsoft, where he gained a broad spectrum of knowledge on different malware types and techniques across the entire kill chain, from destructive malware and ransomware to targeting industrial systems and advanced malware frameworks used for intelligence gathering and espionage. Thomas founded the Unprotect Project in 2015, the most extensive database dedicated to malware evasion techniques, which is a community-driven project and used by thousands of researchers. He has shared his knowledge as a speaker at major conferences, including BlackHat, BSides, and SANS Summits, as well as via his website, Security Break (*https://securitybreak.io*).

APPENDIX A

BUILDING AN ANTI-EVASION ANALYSIS LAB

Building an analysis lab is a critical part of malware analysis, and this is doubly true when it comes to highly evasive and context-aware malware. A well-tuned analysis environment makes the tricky task of analyzing and reversing this type of malware a bit easier. In this chapter, I'll walk you through creating a basic malware analysis lab environment, provide some configuration tips for concealing your hypervisor and virtual machines from malware, and share a few tricks you can use during the analysis process.

Lab Architecture

Malware analysis lab environments contain various virtual machines, software, and other tools that support the analysis process. Lab environments will likely include some or all of the components illustrated in Figure A-1.

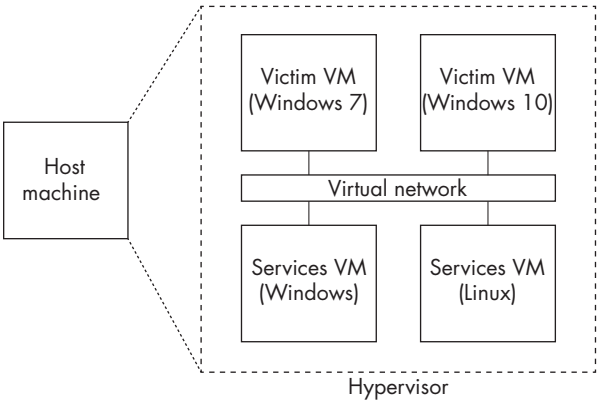


Figure A-1: A typical malware analysis lab environment

Let's go through each in turn.

The Host Machine

Your *host machine* consists of one or more computers that contain and run your malware analysis VMs. It's generally smart to select an operating system for your host that differs from the operating system of the malware you'll primarily be analyzing. For example, in this book, I've focused on Windows malware, so I'd choose Linux or macOS as my host operating system. The reason for this is simple: if the malware you're analyzing were to escape the Windows VM environment (unlikely, but still a risk), having a different operating system on your host would mean that the malware likely wouldn't be able to infect it.

The Hypervisor

The second most important component of a malware analysis lab is the *hypervisor*. Essentially, a hypervisor allocates the host computer's resources (processing power, memory, storage, and so on) to a virtual operating system and its applications (the VM). Hypervisors can run multiple VMs at a time while ensuring that they don't interfere with one another.

Most hypervisors can take a *snapshot*, which is an image of a VM in a particular state and is an important part of malware analysis. After you configure your VMs, remember to take a "clean," preinfection snapshot; this will be your starting point before detonating the malware. You can even take snapshots during malware execution at key points in the analysis process. For example, you may wish to take snapshots of your VM while

debugging a malware executable. If the debugger crashes or the malware is using anti-debugging techniques, you can simply revert to a previous snapshot as necessary. Snapshots can also be reverted to their original state after you've finished your analysis. We'll revisit snapshots later in this chapter.

Two of the most popular hypervisors for both Windows and Linux are VirtualBox and VMware Workstation. We'll return to them in a moment.

Victim Windows VMs

When working with malware that targets Windows, you should dedicate one or more Windows VMs as the "victim" hosts where you'll execute malware and monitor its behaviors. (For malware that targets Linux or macOS, you'd need the equivalent.) Because some malware targets specific versions of Windows, it's wise to keep different configurations of VMs. For example, I use both Windows 7 and Windows 10 VMs, and I keep various versions of software (such as Microsoft Office) installed on them. Note that you should not primarily rely on Windows 7 for malware analysis; as it is now quite dated, it may be missing files and libraries that modern malware depends on!

The malware analysis and research community has very generously provided many handy, free, and open source tools for setting up your victim machines.

Services Windows VMs

As its name suggests, the "services" Windows VM hosts services that may be used to support your malware analysis processes. Examples include Active Directory services (to simulate an AD domain), Server Message Block (SMB) and file-sharing services, chat services (such as IRC), and database servers. If the malware sample you're analyzing is attempting to communicate with other services on the network, it won't hurt to install these services to see how the malware interacts with them. This component of the lab isn't a strict requirement, however, and you may be able to get by without it; it all depends on the capabilities of the malware and what you're trying to achieve in your analysis. You can even simulate some of these services using a network simulation tool such as INetSim, FakeDNS, or FakeNet, as we'll briefly discuss later.

Linux VMs

Even when you're dealing with malware that targets Windows, it's a good idea to have a Linux VM handy. Linux has many command line tools that can save you a lot of time and effort. It can also serve as a network gateway for the Windows victim VMs by monitoring and faking network services. There are even a few prebuilt Linux malware analysis environments. Remnux (*https://remnux.org*) includes nearly all of the tools you'll ever need for malware analysis on Linux. Alternatives to Remnux include SANS SIFT Workstation (*https://www.sans.org/tools/sift-workstation/*) and Tsurugi Linux (*https://tsurugi-linux.org*), but note that these also focus on general forensics and

incident response tasks. Finally, Security Onion (*https://securityonionsolutions.com/software*) is a preconfigured VM image specializing in network traffic analysis and monitoring. It can also be a great addition to your malware analysis toolbox.

Now that you have a basic understanding of what makes up an analysis lab, it's time to build your own!

Building Your Lab

This section walks you through setting up a basic malware analysis lab consisting of a host machine with a hypervisor, a Windows victim VM, and a Linux VM. There are simply too many variations of host OS and hypervisor for me to cover them all, so this lab assumes your host operating system is a variant of Linux, such as Ubuntu, and your hypervisor is either VMware Workstation or Oracle VirtualBox. The following steps should also work for a Windows or macOS host, but keep in mind that there may be slight differences you'll need to adjust for.

Choosing a Hypervisor

Your choice of hypervisor will largely depend on the operating system of your host machine and the resources available to you. Here are some of the most popular hypervisors:

Oracle VirtualBox

VirtualBox (*https://www.virtualbox.org*) is a feature-rich hypervisor that is free for noncommercial use. It includes most of the features that more costly hypervisors have, and it is supported on Windows, Linux, and macOS environments.

VMware Workstation

VMware Workstation (*https://www.vmware.com/products/workstation-pro.html*) has a large set of features and can be installed in a Windows or Linux host environment. It requires you to purchase a license, but VMware provides a free 30-day trial.

VMware Fusion

VMware Fusion (*https://www.vmware.com/products/fusion.html*) is the dedicated VMware hypervisor for macOS. It is very similar to VMware Workstation and also requires a license.

Microsoft Hyper-V

Hyper-V (*https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/*) is a good, free hypervisor for Windows hosts. It can run Windows VMs as well as some Linux-based VMs.

KVM (Kernel-based Virtual Machine)

KVM (*https://linux-kvm.org*) is an open source hypervisor for Linux host environments.

As paid products, VMware Workstation and VMware Fusion have a few additional features that free or open source hypervisors may not have. However, in my experience, VirtualBox is completely suitable for malware analysis, and I don't find myself missing any features while using it.

After you've selected your hypervisor, you'll need to download and install it. For VirtualBox, you can find the latest build of the hypervisor for your operating system, as well as further installation instructions, at *https://www.virtualbox.org/wiki/Downloads*. To download a trial version of VMware Workstation, go to *https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html*.

After installing the hypervisor on your host operating system, you'll need to verify a few settings.

Verifying Hypervisor Network Settings

To implement networking in your VMs later, you need to inspect the VirtualBox hypervisor network settings first. In VirtualBox, navigate to **File ▸ Host Network Manager**.

If no networks are listed here, click **Create** to make one. You can simply use the default settings (set the IPv4 Address to 192.168.56.1, the network mask to 255.255.255.0, and so on), but in the DHCP Server tab, make sure **Enable Server** is checked.

If you don't see a network listed in the VirtualBox Host Network Manager and you get an error such as "Error: VBoxNetAdpCtl: Error while adding new interface: failed to open /dev/vboxnetctl: No such file or directory" when you try to create one, try exiting VirtualBox, executing the following command in a terminal, and then restarting VirtualBox:

```
> sudo modprobe vboxnetadp
```

If you're using the VMware Workstation hypervisor, nothing special is required in terms of network settings, and you can move on to the next step: downloading and installing Windows on your VM.

Obtaining a Windows Image

To build the Windows victim VM, you'll need a copy of Windows 7, 10, or 11, but I'll use Windows 10 as an example going forward since it's my first choice for malware analysis. You may already have a copy and license for Windows lying around. If not, you can get an ISO image file of Windows 10 from *https://www.microsoft.com/en-us/software-download/windows10ISO*. Simply select the version of Windows you want to download, such as Windows 10 (Multi-edition ISO), and select **Confirm**.

Next, you'll select the language of the Windows installation file you want to download, as well as the architecture (either 64-bit or 32-bit). You'll want the 64-bit version unless you'll explicitly be investigating 32-bit malware, which is unlikely. Set the Windows ISO file aside; you'll need it later.

Creating the Windows Victim VM

Now you'll create your Windows VM inside your chosen hypervisor. I'll start with VirtualBox. Later, I'll discuss the same sequence of steps for VMware Workstation.

NOTE *The following instructions include a sequence of menus in the hypervisor. The steps will likely change slightly depending on the version of the hypervisor you're using. If you're missing a certain configuration window or your window appears different from what's described here, the specific configuration will likely show up in another window later in the VM creation process.*

Creating a VM in VirtualBox

If you've selected VirtualBox as your hypervisor, start the program and select **Machine ▸ New**, then specify the name of the VM and the location where it will be stored on disk. Also specify the Type and Version of the operating system you're installing. For our purposes, it should be **Microsoft Windows** and **Windows 10 (64-bit)**, respectively. Click **Next**.

Next, you'll need to configure some basic settings of the VM. Set the Memory Size to 4,096MB (which equates to 4GB) or higher. Evasive malware often uses memory size detection as an anti-VM technique, so it's important to set this value as high as you can (4GB is typically plenty). This also will boost the VM performance. Then, select **Create a Virtual Hard Disk Now** under the Hard Disk settings and click **Next**.

To configure the VM disk image, set a File Size of at least 80GB. Ensure that **VDI** is selected under Hard Disk File Type and that **Dynamically Allocated** is selected under Storage on Physical Hard Disk. Click **Create**.

You should be able to see and select your new VM in the Oracle VM VirtualBox Manager screen, as shown in Figure A-2.

Figure A-2: Your new VirtualBox VM in the Oracle VM VirtualBox Manager

Now we'll cover these same steps in VMware Workstation.

Creating a VM in VMware Workstation

To create a new VM in VMware Workstation, navigate to **File ▸ New Virtual Machine**. You should see the New Virtual Machine Wizard dialog. Under Virtual Machine Configuration, select **Typical (Recommended)** and then click **Next**.

VMware Workstation should prompt you to select how to install the operating system. Choose **Use ISO Image** and browse for the Windows 10 ISO you previously downloaded. Then, click **Next**.

Now you'll need to configure some basic Windows installation settings. Leave the Windows Product Key field empty (unless you have a product key to enter). For Version of Windows to Install, select the appropriate Windows version (for this example, **Windows 10 Pro**). In the Personalize Windows field, enter your username (and optionally a password) for your new Windows installation. Then, click **Next**.

Next, you'll need to specify the name of your new VM as well as the location where it and all of its files should be stored. After configuring these settings, click **Next**.

To configure the VM disk, set the disk size to at least 80GB and then select either **Store Virtual Disk as Single File** or **Split Virtual Disk into Multiple Files**. This choice is strictly based on personal preference. I prefer the latter option because it's easier to transfer smaller VM files to another hard disk or USB drive than it is to transfer one massive file. Once you've made your selection, click **Next**.

Finally, you should see a screen showing an overview of the settings for the new VM. In a bit, we'll customize this VM. For now, be sure to *deselect* **Automatically Power on This Virtual Machine After Creation** and then click **Finish** to create the VM.

Installing Windows in Your VM

Now that you've created your VM in your chosen hypervisor, you're ready to install Windows. To start this installation process, first you'll need to point the VM to the Windows installer image (the ISO file you downloaded previously).

If you're using VMware Workstation and you followed along with the previous instructions, the ISO is already loaded into the VM and ready to go! For VirtualBox, you'll need to right-click your VM and select **Settings** and then **Storage**. Next, select the CD icon both under Storage Devices and in the Optical Drive drop-down menu under Attributes (see Figure A-3), navigate to the Windows ISO file on your disk, and click **OK** to save the configuration.

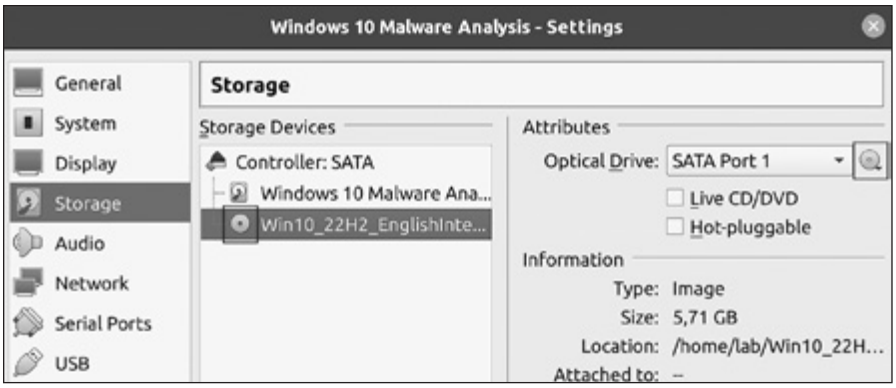


Figure A-3: Adding the Windows installer ISO to the VirtualBox VM

To begin the Windows installation sequence, boot up the VM. To do this in VirtualBox, right-click your VM, mouse over **Start**, and then select **Normal Start**. In VMware Workstation, right-click your VM, mouse over **Power**, and click **Start Up Guest**. The ISO file should load and kick off the Windows installation process.

The Windows installation process takes roughly 20–40 minutes. If you need help completing the Windows 10 installation steps, there are many resources online, such as at *https://answers.microsoft.com*.

Once you've completed the installation, shut down your VM and remove the Windows ISO from it. (Some versions of VirtualBox and VMware Workstation remove it automatically.) For VirtualBox, you can remove the ISO image much like you added it: in the VM's Storage settings, right-click the ISO image and select **Remove Disk from Virtual Drive**. For VMware Workstation, simply make sure **Connect at Power On** is unchecked in the VM's CD/DVD settings.

Tuning VM Settings for Concealment and Isolation

Next, you'll do some basic configuration and tuning to help limit the VM's footprint, making it more difficult for evasive malware to detect that it is running inside a VM. Isolating the VM from the host operating system is also a safety measure to better protect the host during malware analysis. These settings are typically very easy to implement and quite effective, so don't disregard them.

Memory and Processors

To address malware trying to detect a VM through CPU and memory enumeration, set your VM memory as high as possible (4GB at minimum) and use at least two processors. This may trick the malware into thinking it's executing in a non-VM environment.

In VirtualBox, to modify the memory, go to **Settings ▸ System ▸ Motherboard**. To modify the CPU settings, navigate to **Settings ▸ System ▸ Processor**.

To access the memory settings in VMware, go to **Settings ▸ Memory**. For the CPU settings, go to **Settings ▸ Processors**.

Another benefit of assigning more CPU power and memory to your analysis VMs is performance. The "beefier" your analysis VMs, the better they perform during malware analysis, especially given that some malware analysis tools use a lot of system resources. Keep in mind that evasive malware uses several techniques to interfere with analysis sandboxes and VMs based on system performance and resources, such as API hammering (covered in Chapter 8).

Hard Disk Size

Checking the hard disk size is one of the oldest, simplest, and most common techniques malware uses to detect a VM. VMs are notorious for having small hard drives, so assign your virtual disk drive at least 60GB of space. Typically, I assign 80GB or more. If you followed the VirtualBox and VMware Workstation VM creation instructions earlier in this chapter, you've already done this step.

To check your virtual disk drive storage space in VirtualBox, go to **Settings ▸ Storage**. In VMware, go to **Settings ▸ Hard Disk**.

You can extend the hard disk size of a VM retroactively, but it's generally best to configure hard disk size when you create the VM.

Display Settings and Acceleration

Features supporting *3D acceleration* add performance enhancements to a VM, but they may expose the hypervisor to certain malware. To protect against detection, disable these options. In VirtualBox, navigate to **Settings ▸ Display** and on the Screen tab, make sure **Enable 3D Acceleration** isn't selected.

In VMware, navigate to **Settings ▸ Display** and deselect **Accelerate 3D Graphics**.

USB Controller Settings

Some malware attempts to enumerate the USB controller on the system. If the system is using outdated USB drivers (such as version 1.0 or 2.0, as opposed to the newer 3.0 drivers), the malware might assume it's running in an analysis machine. To configure this setting, in VirtualBox go to **Settings ▸ USB**, and in VMware Workstation go to **Settings ▸ USB Controller**.

Network Adapter Configurations

A critical part of malware analysis in a VM is understanding and properly utilizing the right VM network configuration for the task at hand. There

are different types of network configurations you can assign to your analysis lab VMs, but these are some of the most important modes:

Not Attached

The Not Attached mode in VirtualBox (for VMware Workstation, this setting is a checkbox labeled Connect at Power On, which must be unchecked) essentially switches off networking for the VM. The VM will be completely isolated from any networks, unable to communicate with other VMs, the local host's network, or the internet. This is the safest option for analyzing malware. However, modern evasive malware expects some sort of network connection, so it may not execute fully (or at all) while the VM is in this mode. For this reason, I won't discuss this mode further in this chapter.

Host-Only

The Host-Only connection is a private network that is shared with the host operating system. In this configuration, the VM won't have access to the internet. It will, however, have network access to the host and other VMs running on the host. This option is a good middle ground between safety and effectiveness, especially when you're using another VM configured as a network gateway, as we'll explore later in this chapter.

Bridged and NAT

In both Bridged and Network Address Translation (NAT) modes, the VM is connected to the host's local network, allowing it to access the internet and other network resources. In Bridged mode, the VM has its own IP address separate from the host. In NAT mode, the VM shares the host's IP address and can't be reached directly from the local network. The most important point here is that the VM (and any running malware!) is able to reach out to the internet. NAT mode provides a bit of extra security, so I use this mode if I need my VM to have internet access.

As a rule of thumb, I nearly always keep my analysis VMs in Host-Only mode. I use a Linux VM as a network gateway for the Windows victim VM to fake an internet connection, which we'll talk about more later. However, as described in Chapter 6, an increasingly common anti-VM and anti-sandbox technique is for malware to attempt to contact a remote server to determine whether the VM is connected to the internet. Some malware might also download modules or payloads from an attacker-controlled server, and you can miss this activity if the analysis environment is isolated. In these special cases, it makes sense to put your VM in NAT or Bridged mode. Just be cognizant of the risks of connecting live malware to the internet. For example, the malware may be able to steal data from your VM (such as from your clipboard or any virtual shared drives) or even add your VM to a botnet, in which case your VM may be used without your permission to commit crime.

To configure your VM's network adapter in VirtualBox, navigate to **Settings ▸ Network**, and on the Adapter 1 tab, make sure **Enable Network Adapter** is checked. Then, in the **Attached To** drop-down menu, change the VM network adapter to **Host-only Adapter**, **NAT**, or **Bridged**,

depending on your needs (see Figure A-4). For now, select **NAT** or **Bridged** mode, as you'll need access to the internet in a moment.

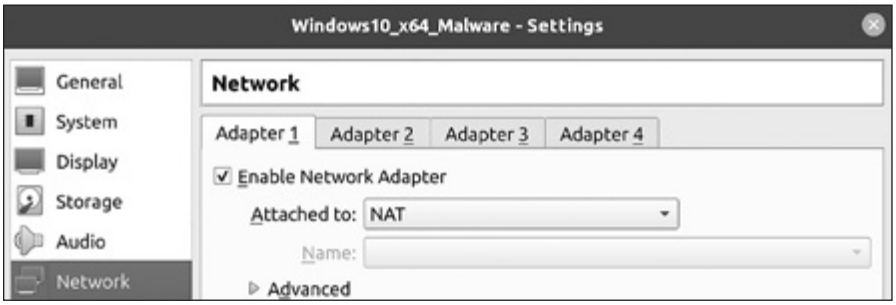


Figure A-4: Configuring your VM's network adapter in VirtualBox

If you're unable to set the network adapter to NAT, you may need to first configure a NAT network in VirtualBox. To do this, navigate to **File ▸ Preferences ▸ Network** and click **+**.

To configure the network adapter in VMware Workstation, navigate to **VM Settings ▸ Hardware ▸ Network Adapter** and select the network connection type you require (see Figure A-5). For now, select **NAT** or **Bridged** mode.

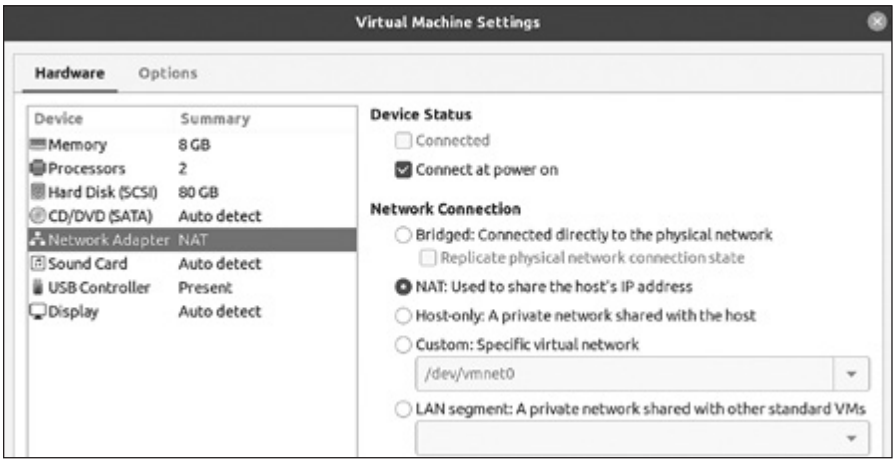


Figure A-5: Configuring your VM's network adapter in VMware Workstation

MAC Addresses

Also listed under the network configuration options are MAC address settings. Hypervisors often use standard MAC address ranges for their virtual network adapters. For example, VirtualBox may use the MAC address prefixes 00:00:7D, 00:01:5D, 00:0F:4B, 00:10:E0, 00:14:4F, 00:21:28, 00:21:F6, 08:00:27, or 52:54:00. VMware may use the prefixes 00:05:69, 00:0C:29, 00:1C:14, or 00:50:56.

To circumvent MAC address–based VM detection, simply change the default MAC address of your VM to a different prefix.

NOTE *For a fairly complete list of MAC address prefixes you can use, see* https://gist.github.com/aallan/b4bb86db86079509e6159810ae9bd3e4. *Ideally, select a MAC address that corresponds to a well-known network adapter manufacturer.*

To change your MAC address in VirtualBox, navigate to **Settings ▸ Network**. On the Adapter 1 tab, click the arrow next to Advanced and then enter the new address in the MAC Address field (see Figure A-6).

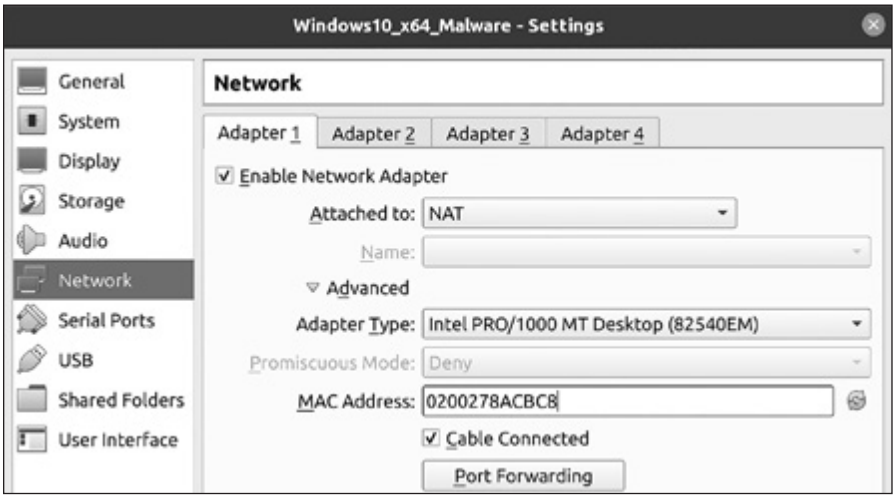


Figure A-6: Network adapter settings in VirtualBox

For VMware, navigate to **Settings ▸ Network Adapter ▸ Advanced** and enter the new address in the MAC Address field (Figure A-7).

Figure A-7: Network adapter settings in VMware

In both VirtualBox and VMware, you can generate a random MAC address simply by clicking the refresh symbol (VirtualBox) or **Generate** (VMware), next to the MAC Address field. The generated random addresses are still within the normal hypervisor address range, however, so it's best to set this manually with a new prefix in order to avoid detection.

Clipboard and Drag-and-Drop Settings

Some hypervisors (including VMware Workstation and VirtualBox) allow clipboard sharing between host and guest systems. This means you can copy data from your host machine and paste it into your guest VM, and vice versa. This feature may be convenient, but it carries some risk. When clipboard sharing is enabled, any data in your host system's clipboard is theoretically available to your guest VM. If you copy sensitive data (such as a password) into your clipboard on your host machine, malware running in the guest VM may be able to access it. Likewise, the malware could use the clipboard to write data to the host system or exploit potential vulnerabilities in the hypervisor. This scenario is unlikely but still possible.

Drag-and-drop features allow you to drag (copy) files from your host machine to your guest VM, and vice versa. Much like clipboard sharing, this could expose your host machine to more risk than necessary, depending on the nature of the malware you're analyzing. Enable these features only if absolutely required.

To turn off clipboard and file drag-and-drop settings in VirtualBox, navigate to **Settings ▸ General ▸ Advanced** and select **Disabled** in the **Shared Clipboard** and **Drag'n'Drop** drop-down menus (see Figure A-8).

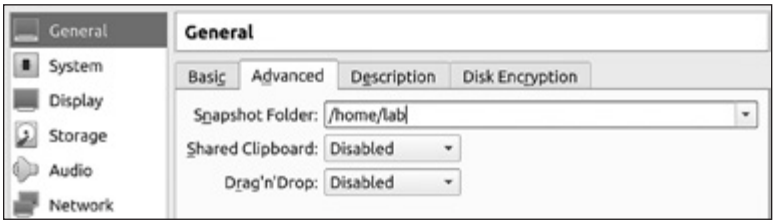


Figure A-8: The clipboard and drag-and-drop settings in VirtualBox

In VMware, navigate to **Settings ▸ Options ▸ Guest Isolation**, as shown in Figure A-9.

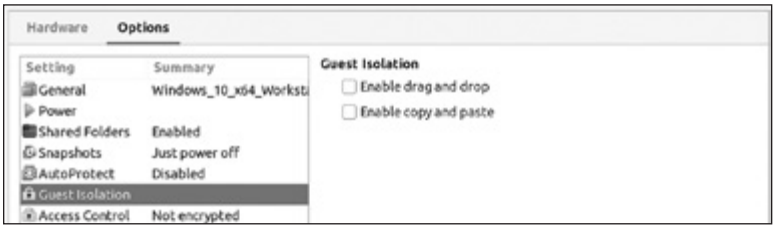


Figure A-9: The clipboard and drag-and-drop settings in VMware

In this menu, disable drag-and-drop and clipboard sharing by deselecting the **Enable drag and drop** and **Enable copy and paste** options.

Shared Folders

Shared folders allow easy sharing of files from guest to host operating system. Keep in mind, however, that malware will also have access to whatever is in your shared folder. (I learned this the hard way.) Enable shared folders only if necessary; if you must use them, set them to "read only" as a minimum precaution.

You can find shared folder settings in VirtualBox (see Figure A-10) by going to **Settings ▸ Shared Folders**.

Figure A-10: Shared folder settings in VirtualBox

To add a shared folder in VirtualBox, click the icon of a folder with the plus sign (+) on the right side of the menu. You can also edit a shared folder configuration by double-clicking the shared folder under Machine Folders. To remove a shared folder, click the icon of a folder with the (X) sign.

In VMware, shared folder settings are also under Settings ▸ Shared Folders (see Figure A-11).

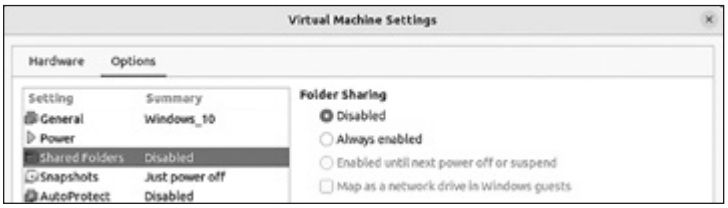


Figure A-11: Shared folder settings in VMware

You can add and edit shared folders from this menu. To disable shared folders, select **Disabled** under **Folder Sharing**.

NOTE *Clipboard sharing, drag-and-drop settings, and shared folders are functional only if you have the optional VirtualBox Guest Additions or VMware Tools installed in your VM. We'll discuss these tools later in this chapter.*

Installing Windows Malware Analysis Tools

You should now have a functioning Windows VM that is already tuned to be quite resistant to many basic VM detection and evasion techniques. This alone isn't sufficient for your malware analysis journey, however; you'll also need analysis tools. I recommend downloading and installing FLARE-VM (*https://www.mandiant.com/resources/blog/flare-vm-the-windows-malware*), a fully configured malware analysis environment from Mandiant. It includes a series of scripts that prepares Windows for malware analysis tasks by downloading and installing many useful tools. It's not a requirement to install FLARE-VM, but it can save you a lot of time. To download and install FLARE-VM, boot up your Windows VM and carefully follow the installation steps from the *README* at *https://github.com/mandiant/flare-vm*.

If you choose not to install FLARE-VM, you should at least take the following measures to prepare your malware analysis environment:

1. **Disable Windows updates.** Typically you won't want your malware analysis environment to receive regular Windows updates, so it's a good idea to disable them. For instructions, see *https://www.windowscentral.com/how-stop-updates-installing-automatically-windows-10*. Keep in mind, however, that if you disable Windows updates, you might miss any attempts by malware to exploit versions of the operating system or application software you do not have installed.
2. **Disable Windows tamper protection.** Disabling Windows tamper protection is a necessary step before you can disable Microsoft Defender (described next). You can disable tamper protection in the **Windows Security ▸ Virus and Threat Protection** settings. For more information on disabling tamper protection, see *https://support.microsoft.com/en-us/windows/prevent-changes-to-security-settings-with-tamper-protection-31d51aaa-645d-408e-6ce7-8d7f8e593f87*.
3. **Disable Microsoft Defender.** Disabling Defender prevents anti-malware software from interfering with your malware analysis environment. Learn how to disable it at *https://www.windowscentral.com/how-permanently-disable-windows-defender-windows-10*.

If you chose not to install FLARE-VM, you'll need to manually install your tools. Table A-1 summarizes the tools I use in my environment, many of which I've mentioned throughout this book, and what they do.

Table A-1: Windows-Based Malware Analysis Tools

Tool type	Purpose	Example(s)
Advanced task manager	Interact with running processes and malware	Process Hacker *https://processhacker.sourceforge.io*
Debugger	Dynamically analyze malicious code	x64dbg *https://github.com/x64dbg/x64dbg*
Disassembler	Reverse engineer malware	IDA Pro *https://hex-rays.com/ida-free/* Ghidra *https://github.com/NationalSecurityAgency/ghidra*
File detector	Detect various file types, identify packers and obfuscators, and more	Detect It Easy *https://github.com/horsicq/DIE-engine/releases*
Hex editor	View and modify binary data	HxD *https://mh-nexus.de/en/hxd/*
Network monitoring tool	Monitor and inspect the network interactions of a malware sample	Wireshark *https://www.wireshark.org*
PE analyzer	Get an overview of PE-based malware	PEStudio *https://www.winitor.com/download*
Process monitor	Monitor malware processes and their interactions with the operating system	Procmon *https://learn.microsoft.com/en-us/sysinternals/downloads/procmon*
Registry and baseline comparison utility	Compare a system state to a baseline state after detonating malware	Regshot *https://sourceforge.net/projects/regshot/*
Web proxy	Intercept and monitor web requests initiated by the malware	Fiddler *https://www.telerik.com/fiddler*

Depending on what kind of malware you're analyzing, you may need other tools and software. For example, if you're dealing with Excel and Word files, you'll have to install Microsoft Office; to analyze the behaviors of malicious PDFs, you'll probably need Adobe Acrobat; and if you're investigating .NET executables, you'll need the .NET framework and its associated libraries. Be sure to identify, install, and configure the software required for detonating the files you'll be investigating. Note that FLARE-VM may not contain all of the tools you'll need, so you'll have to manually install them.

Installing VM Tools

VM tools is a generic term for hypervisor software that can be installed inside a guest VM. In VirtualBox, this tool set is called Guest Additions; in VMware Workstation, it's VMware Tools. This software increases the usability and performance of the VM, and it also adds helpful features such as shared folders and clipboard sharing. Unfortunately, these tools also introduce anomalies, such as processes and driver files, that malware can use to detect the hypervisor.

Even with the risks, these tools add convenient functionality and extra performance for the analysis VM. I take a twofold approach: I have one Windows VM without the VM tools installed and one VM with them installed. I use my VM with the tools installed as my primary analysis environment. If the malware I'm investigating is particularly problematic in its evasion and VM detection capabilities, I switch to the toolless VM. This works well for me, and it likely will for you also.

Another option is to uninstall the VM tools using the Windows software uninstaller prior to detonating problematic malware. And finally, there are two tools, VBoxCloak and VMwareCloak, that have an option to clean up some of the files and *cruft* (unwanted processes and artifacts) left from installing VM tools. We'll look at them later in the chapter.

To install Guest Additions in a VirtualBox VM, start the VM and, once Windows has booted, go to **Devices ▸ Insert Guest Additions CD Image**.

The Guest Additions installer files are now accessible in the virtual CD drive of your VM. In my case, this is the *D:* drive. Double-click the **VBoxWindowsAdditions.exe** executable to start the Guest Additions installer (see Figure A-12). Don't forget to reboot the VM after installing.

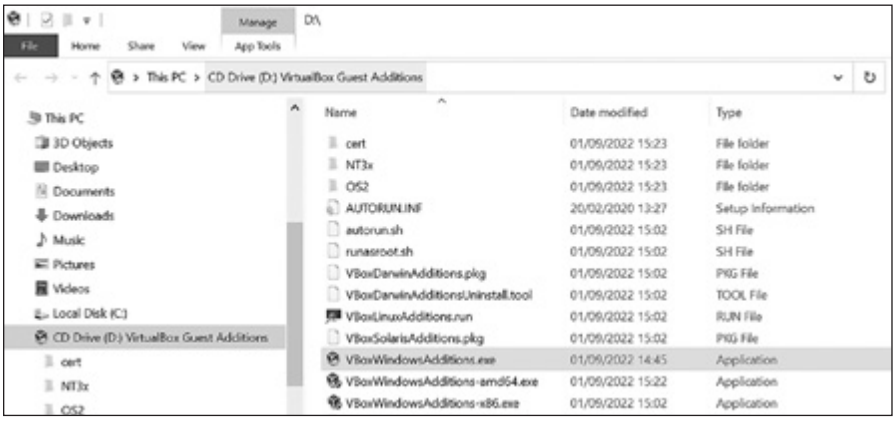


Figure A-12: Installing Guest Additions

For more information or help with the installation process, see the VirtualBox Guest Additions documentation at *https://www.virtualbox.org/manual/ch04.html*.

For newer versions of VMware Workstation, VMware Tools is often installed automatically. If you need to install it manually, the process is nearly identical to that for VirtualBox. In the VMware Workstation VM, navigate to **VM ▸ Install VMware Tools**. (This option appears as **Reinstall VMware Tools** in my case, since I already have the tools installed, as shown in Figure A-13.) As with VirtualBox Guest Additions, you'll need to reboot the VM after installation.

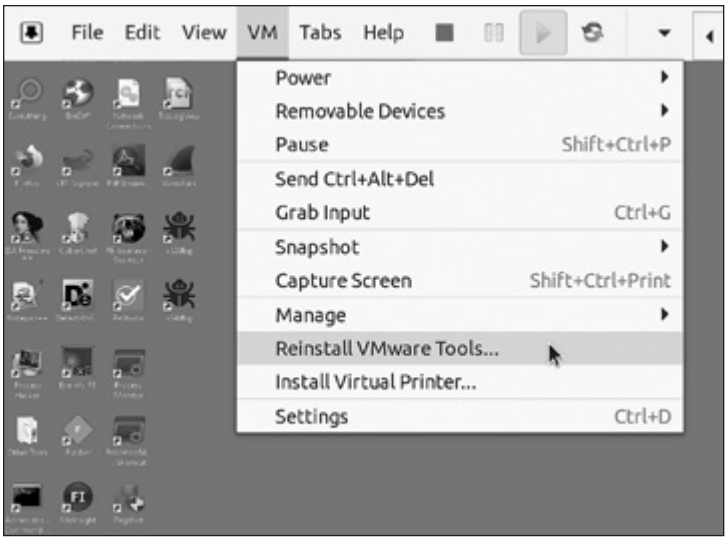


Figure A-13: Installing VMware Tools

Now we'll take a short break from our Windows VM to discuss how to set up a Linux VM.

Installing and Configuring a Linux VM

One of the primary benefits of having a Linux VM in your lab is that it can act as a lightweight gateway for the Windows VM. As you detonate malware in your Windows VM, your Linux VM can intercept network traffic for later analysis, and it can even fake network and internet services, as you'll see later. I use Remnux in my lab, so that's what I'll cover in this guide. Remnux, as mentioned earlier in this chapter, is a prepackaged, fully capable Linux malware analysis environment. It has most of the tools you'll ever need for static analysis of malicious files and code, as well as some options for dynamic analysis (such as code emulation tools). You can download Remnux at *https://docs.remnux.org/install-distro/get-virtual-appliance*. Simply select the appliance you need (either VirtualBox or VMware) and download and configure the VM according to the instructions provided. When you're finished, you should have a working Remnux VM. Don't forget to update Remnux using the following commands:

```
> remnux upgrade
> remnux update
```

NOTE *Before updating Remnux to its latest version, you'll need to give Remnux internet access, so be sure to set its network adapter to NAT or Bridged mode before using it. You can set it back to Host-Only after the updates are completed.*

Manually Installing Linux VM Tools

Using Remnux is optional, and you may choose to configure your own Linux VM from scratch instead. If you do, you'll need to install your malware analysis tools yourself. Table A-2 lists some Linux tools I consider essential for malware analysis. These tools are all preinstalled and configured in Remnux. Note that some of these tools are also included in FLARE-VM for Windows.

Table A-2: Linux Malware Analysis Tools

Tool	Purpose
Base64dump *https://github.com/DidierStevens/DidierStevensSuite/blob/master/base64dump.py*	Identifies and extracts Base64-encoded data from a file
Binwalk *https://github.com/ReFirmLabs/binwalk*	Analyzes binary images and extracts embedded files (such as malware that uses steganography techniques)
CAPA *https://github.com/mandiant/capa*	Scans for and detects suspicious signatures in executable files, such as potential evasion and obfuscation techniques
ExifTool *https://exiftool.org*	Identifies file types and allows you to view and edit their metadata
FakeDNS *https://github.com/SocialExploits/fakedns/blob/main/fakedns.py*	Responds to DNS queries and simulates a DNS service
FLOSS *https://github.com/mandiant/flare-floss*	Extracts encoded and obfuscated strings from a PE file
INetSim (Internet Services Simulation Suite) *https://www.INetSim.org*	Simulates different network services (such as DNS, FTP, and HTTP)
Speakeasy *https://github.com/mandiant/speakeasy*	Emulates executable code and shellcode
XORSearch *https://blog.didierstevens.com/programs/xorsearch/*	Scans a file for strings encoded and obfuscated in various formats (such as XOR or ROL)
Yara *https://github.com/Yara-Rules/rules*	Identifies and classifies malware

This section has only scratched the surface of the useful tools available on Remnux and for Linux-based analysis environments; there are also malicious document analysis tools, emulation tools, and memory forensics tools, but a full discussion of them is beyond the scope of this book.

Configuring and Verifying Network Settings

You've nearly completed the setup of your malware analysis lab, but there are a few more steps. Before proceeding, make sure the network adapters for *both* your Windows VM and Remnux VM are set to **Host-Only**. This is very important for the next steps you'll take to finalize the lab setup.

Next, you'll need to get some network adapter information from the Remnux VM. Execute the **`ifconfig`** command in a terminal in Remnux. Figure A-14 shows some example output.

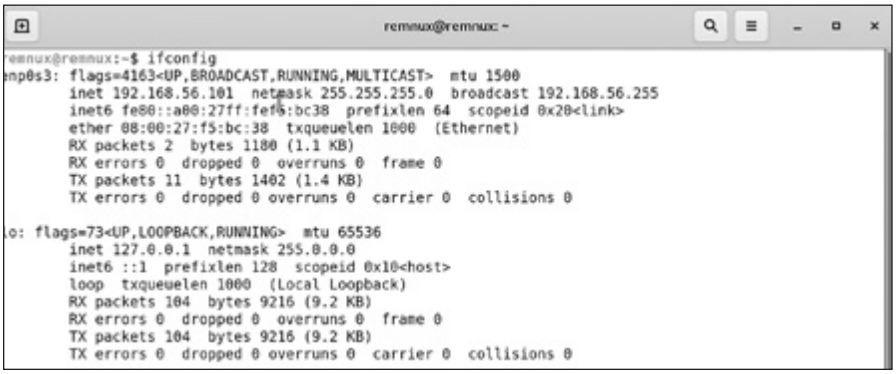


Figure A-14: Obtaining the Remnux operating system network configuration

The first entry listed in this output is what we care about. The `inet` (IP) address of this VM is `192.168.56.101`, and the `netmask` is `255.255.255.0`. Your results may be different depending on your specific Remnux configuration and whether you're using VirtualBox or VMware. Jot down these values, as you'll need them in a minute.

Return to your Windows VM and navigate to the Windows network settings from the Start menu. Set the IP address of your Windows VM to the same subnet as your Remnux VM. (For example, if your Remnux VM IP address is `192.168.56.101`, you might set your Windows VM IP address to `192.168.56.102`.) If the netmask of your Remnux VM is `255.255.255.0` (the default), enter **`24`** in the Subnet Prefix Length field. For the Gateway address, enter the Remnux VM's IP address (since Remnux will be acting as the gateway for the Windows VM), and enter it again for the Preferred DNS address. Figure A-15 shows how this configuration looks in Windows 10.

Figure A-15: Configuring Windows VM IP settings

Click **Save** to set the configuration. You may need to reboot your Windows VM.

Now you'll test the connection between the Remnux VM and Windows VM. Make sure both VMs are powered on, and execute a **`ping`** command to the Remnux IP in your Windows VM, as shown in Figure A-16.

```
C:\Users>ping 192.168.56.101

Pinging 192.168.56.101 with 32 bytes of data:
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
```

Figure A-16: Testing the lab network configuration

This command should return a `Reply`, similar to the output shown here. If not, you'll have a bit of troubleshooting to do. For starters, confirm that the Remnux VM is powered on, that the Windows and Remnux network adapters are set to Host-Only in your hypervisor, and that your Windows IP address configuration is correct.

There is one last step to finalize your new lab environment: take snapshots of the VMs.

Taking and Restoring VM Snapshots

As mentioned earlier in this chapter, snapshots allow you to save a VM in a certain state; in this case, that will be the pristine, clean state before the Windows VM is infected with any malware. First, shut down your Windows and Remnux VMs by initiating a normal shutdown within the operating system.

To take a snapshot in VirtualBox, select your Windows VM and go to **Snapshot ▸ Take**. Be sure to name the snapshot something that makes sense to you (such as "Windows Malware Analysis – Clean"). Repeat this process for your Remnux VM.

To take snapshots in VMware Workstation, right-click the Windows VM and select **Snapshot ▸ Take Snapshot**. Again, name the snapshot something intuitive, and repeat these steps for Remnux.

To revert to a snapshot (for example, after you detonate and analyze a malware sample), you'll need to access the hypervisor's snapshot manager. In VirtualBox, you can access this by selecting a virtual machine and then navigating to **Machine ▸ Tools ▸ Snapshots**. The snapshots are listed in the right window pane, under Name. Figure A-17 shows a list of snapshots for my VM. (I called the first snapshot in the list "BASE – 08 Aug 23 – Pristine Windows 10," but you should name your snapshots whatever makes sense to you.) To restore a previous snapshot, right-click it and select **Restore**.

Figure A-17: The VirtualBox Snapshot Manager

In VMware Workstation, the Snapshot Manager is a bit more hidden away. To access it, right-click your VM and select **Snapshots ▸ Snapshot Manager**. You'll see a tree graph view of all your snapshots, as shown in Figure A-18. Simply right-click a snapshot and select **Restore**.

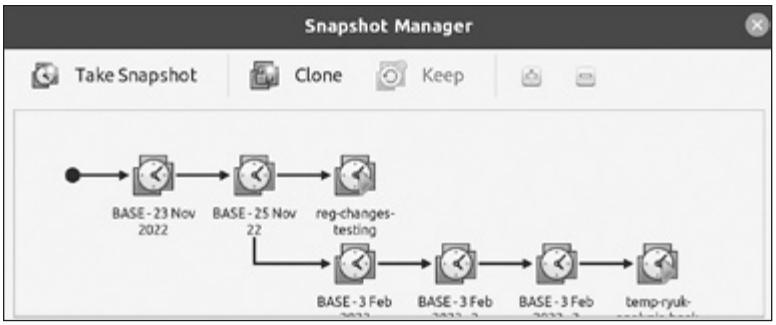


Figure A-18: The VMware Snapshot Manager

Snapshots are powerful tools not only for restoring a VM to a pristine state but also for preventing analysis headaches. For example, one of my strategies is taking snapshots of the VM at certain phases of debugging. Sometimes a debugger will crash during analysis, or the malware may execute code covertly to "escape" the debugger. Reverting to a previous debugging snapshot lets me avoid having to start all over again.

If you followed the previous steps, you should have a working malware analysis lab. You have a Windows victim VM where malware can be safely detonated and a Linux VM for simulating network services and capturing network traffic. You've also configured your Windows VM virtual hardware for robustness against malware trying to detect it. Now let's turn to how you can configure your operating system to further conceal the Windows VM.

Windows Configurations for Concealment

There are several optional Windows settings and tips that you can apply to your Windows VM to help hide it from context-aware malware. Most of these aren't exactly advanced, and some might even seem a bit absurd, but incorporating them can make your analysis system more resilient and discreet.

Registry Data

As you learned in Chapter 4, the Windows registry contains a wealth of information related to the operating system and hardware that the malware could query to detect a hypervisor. Fortunately, you can modify many of these registry keys, values, and data to circumvent detection. You can do this directly in the Windows Registry Editor (RegEdit) or by using PowerShell. For example, run the following PowerShell command to modify a registry key's value:

```
PS C:\> Set-ItemProperty -Path Registry Path -Name Name of Value -Value Registry Data
```

To rename the value inside `BIOSProductName` to `Fake BIOS`, execute the following command:

```
PS C:\> Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\
SystemInformation" -Name "BIOSVersion" -Value "Fake BIOS"
```

Many registry keys that could be of interest to malware are updated within different iterations of Windows and hypervisor versions and patches, so it's not feasible to list them all here. Instead, I've created a simple script in PowerShell that scrubs the registry to hide some of these indicators and also accomplishes a number of other VM concealment tasks that I'll discuss throughout this section. You can find the VirtualBox version of the script, VBoxCloak, at *https://github.com/d4rksystem/VBoxCloak*, and the VMware version, VMwareCloak, at *https://github.com/d4rksystem/VMwareCloak*.

Hostname and Domain Name

Since some advanced malware enumerates the analysis environment's hostname, domain name, and user account information to determine if it's running in a VM, it's wise to set these values to something innocuous. The malware might look for strings such as `sandbox`, `virus`, `malware`, `VMware`, `virtualbox`, `test`, or `cuckoo`, for example. Ideally, you should set your system hostname and primary user account name when you install and configure the system, but you can also change these before detonating the malware.

To change the system hostname using PowerShell, use the following command:

```
PS C:\> Rename-Computer -NewName "new hostname"
```

To change the local user account name, use this command:

```
PS C:\> Rename-LocalUser -Name "current local username" -NewName "new local username"
```

You'll need to reboot the VM for these changes to take effect. Additionally, some malware (like certain variants of infostealers and ransomware) tests to see if a system is part of a corporate domain (or, in the case of more targeted malware, a *specific* corporate domain) before infecting it. Adding a fake domain name to your system can help you avoid detection in these cases. You can do this by creating an actual domain (using a domain controller) or, more simply, by issuing the following PowerShell command, which will "add" the system to the domain *corp.com*:

```
PS C:\> Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\Tcpip\
Parameters\" -Name "Domain" -Value "corp.com" -Force
```

Keep in mind that this registry change doesn't add the Windows system to a real domain; it simply changes one configuration setting that malware may query. I've created a short script that automatically changes the system's hostname and local user account name and then adds the VM to a

fake domain using this registry trick. You can find this script at *https://github.com/d4rksystem/hostname-changer*. However, to fully simulate a domain environment, a better approach is to set up a real domain controller in your lab.

Additional Tips and Tricks

Here are some additional configuration tips and tricks that may prove valuable in certain circumstances:

Renaming analysis tools and installing them in nonstandard locations

Some crafty malware looks for running analysis tools such as Wireshark or Procmon. Simply renaming the tool's executable files (for example, from *wireshark.exe* to *krahseriw.exe*) before launching them can thwart this detection technique. (Note that renaming an executable in this way may break the tool's functionality.) It can also be useful to install your tools in nondefault locations.

Adding decoy files

Malware may inspect the victim system's *Desktop* or *Documents* directories and infect the system only if there are files and documents there. It never hurts to add a few fake documents (*invoice.doc*, *passwords.txt*, and the like) to these directories to simulate a normal Windows user.

Activating the mouse

Context-aware malware might sleep until the mouse moves or a certain mouse button is pressed. Moving the mouse manually and clicking can help circumvent these simple mouse detection techniques. You can even automate mouse activities inside your VMs and sandboxes using a Python library like PyAutoGUI (*https://pyautogui.readthedocs.io/en/latest/*).

Changing the malware filename and path

Malware sometimes checks its running location to see its filename and path. Some malware sandboxes automatically name a malware file by its MD5 or SHA-1 hash, which can be a dead giveaway. To conceal your VM, it's best to name the malware file something random and not include words like *malware*, *virus*, and *lab* in the filename or run path. Some malware also checks its run path to ensure it's running from a directory the author intended and not from *Documents*, *Desktop*, and so on. Sometimes the malware may even verify it still has its original filename.

The tool exiftool (which I briefly mentioned earlier in this chapter), as well as many other PEStudio-type tools, allows you to view the Original File Name field of an executable file, which may be a hint into what the malicious file was originally named. In the following code, you can see the output of exiftool and an executable file's original name:

```
> exiftool malware.exe
--snip--
File Version                : 6.0.7.2527
```

```
Internal Name                : RealOne Player
Legal Copyright              : Copyright © 2001-2002
Original File Name           : player.exe
Product Name                 : RealOne Player
--snip--
```

You can even often find clues in the strings of the malware file that indicate its original filename, as in this example (*epmntdrv.sys*):

```
> strings evil.bin
--snip--
Invalid parameter passed to C runtime function.
h:\projectarea\OO_source\mod.windiskaccessdriver\epmntdrv.sys
ExAllocatePoolWithTag
--snip--
```

Adding system uptime

Malware may check how long the system has been booted before fully executing, or it may not run if the system has an insufficient system uptime. Waiting a few minutes after booting your analysis VM before detonating the malware may trick it into executing. Better yet, prior to infecting the VM, let it run for 20 minutes and then take a snapshot of the system. You can later revert to this snapshot and the VM will already be in a state that has been running, ready for malware detonation.

Mimicking your organization or the malware target

Before detonating a targeted malware sample, you can configure the environment to match the malware's target as closely as possible. For example, adding the machine to a fake but realistic domain may help extract behaviors from the malware that you'd otherwise not see.

Part II discussed VM artifacts and detection in detail, so refer to those chapters for more information to help you conceal your analysis VMs.

Advanced VM and Hypervisor Hardening

In addition to VM hardware and guest operating system configurations, you can apply so-called hardening techniques to your VMs and hypervisor. *Hardening* involves configuring the more advanced settings of the VM or hypervisor or even patching the hypervisor directly. This section discusses some of these tools and techniques for VMware Workstation and VirtualBox.

NOTE *These techniques are included in the book for completeness. Depending on your host operating system, guest operating system, and hypervisor version, they may be ineffective or even cause stability or performance issues in your VMs, so use them at your own risk.*

Hardening VMware

Each VMware VM has a VMX (*.vmx*) file that contains the machine's configurations. You can modify this file to configure some of the more advanced options for your VM. The VMX file resides in the VM's home directory. (For example, on my Linux host, it's located at */home/<user>/VMware/<vm_name>/<vm_name>.vmx*.) VMware VMs have notable system manufacturer and model strings that can raise flags for malware. A default Windows VM running in VMware looks like this:

```
System Manufacturer:        VMware, Inc.
System Model:               VMware Virtual Platform
```

Adding this simple line to the VMX file may help conceal your VM by mirroring the host's system information in the guest VM:

```
SMBIOS.reflectHost = "True"
```

Malware may also attempt to detect the disk drive model of your VMware VM, which will be quite generic if the hardware is virtualized. To circumvent this, add these lines to your VMX file (you can replace "Samsung" with anything you'd like):

```
scsi0:0.productID = "Samsung SSD"
scsi0:0.vendorID = "Samsung"
```

To fend off some `cpuid`- and `rdtsc`-based VM detection techniques, add these lines to your VMX file:

```
hypervisor.cpuid.v0 = "FALSE"
monitor_control.virtual_rdtsc = "FALSE"
```

NOTE *As Chapter 7 discussed, `cpuid` can be used to detect whether a machine's processor is virtualized, and `rdtsc` can be used to perform processor timing analysis.*

These are simple changes, but as previously mentioned, your mileage may vary depending on your operating systems and versions. For example, I was unable to reflect my host system information to my VM using the `SMBIOS.reflectHost` trick with a host system running Linux Ubuntu 20 and a Windows 10 guest VM. However, it worked on a Windows 10 host with a Windows 10 guest.

Here are some other known VMX configurations you can add to your VMs:

```
SMBIOS.noOEMStrings = "TRUE"
serialNumber.reflectHost = "TRUE"
hw.model.reflectHost = "TRUE"
board-id.reflectHost = "TRUE"
```

```
monitor_control.restrict_backdoor = "TRUE"
monitor_control.disable_directexec = "TRUE"
monitor_control.disable_reloc = "TRUE"
monitor_control.disable_btinout = "TRUE"
monitor_control.disable_btmemspace = "TRUE"
monitor_control.disable_btpriv = "TRUE"
monitor_control.disable_btseg = "TRUE"
monitor_control.disable_chksimd = "TRUE"
monitor_control.disable_ntreloc = "TRUE"
monitor_control.disable_selfmod = "TRUE"
```

The first group in this configuration may help hide the VM by reflecting host information to the guest, rather than using the default VMware strings. The second group pertains to how binary code is emulated in the guest VM, how the VM interacts with the physical processor, and other functions. It may help circumvent malware that uses these settings to detect a VM. Many of these configurations are undocumented by VMware, but a few notable projects seek to identify and elaborate on them. For example, check out the research from Tom Liston and Ed Skoudis in their presentation "On the Cutting Edge: Thwarting Virtual Machine Detection," at *https://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf*, and read about `monitor control` in the list of advanced parameters at *http://sanbarrow.com/vmx/vmx-advanced.html*. There is also an older tool called VmwareHardenedLoader (*https://github.com/hzqst/VmwareHardenedLoader*), which is a set of scripts and configurations that performs many of the aforementioned changes, plus some others.

Hardening VirtualBox

VirtualBox is a bit trickier to tune; it doesn't have the equivalent of a VMX file. Instead, you're forced to use VBoxManage, an application for Windows and Linux that's specifically designed for making configuration changes to VirtualBox VMs. For example, to prevent some `rdtsc` VM detection techniques, you can configure your VM by running the following commands in the command line:

```
> VBoxManage setextradata "vm_name" VBoxInternal/TM/TSCMode RealTSCOffset
> VBoxManage setextradata "vm_name" VBoxInternal/CPUM/SSE4.1 1
> VBoxManage setextradata "vm_name" VBoxInternal/CPUM/SSE4.2 1
```

Some configurations in VirtualBox are complicated in comparison to VMware and (at the time of this writing) are surprisingly difficult to find information on. Fortunately, since VirtualBox is open source, some members of the community have written hardeners for VirtualBox and its VMs. Much like for VMware, there is also VBoxHardenedLoader (*https://github.com/hfiref0x/VboxHardenedLoader*), which you may want to look into.

The main problem with some of these hardeners is that they can break with different iterations of the hypervisor, so they must be modified for each new version of VirtualBox. As with any tool or configuration mentioned in this chapter, your success depends on your specific lab environment.

Stress-Testing Your VM

Prior to detonating malware, particularly potentially evasive malware, it can be helpful to stress-test your Windows analysis VM against detection techniques by using a tool such as Pafish (*https://github.com/a0rtega/Pafish*), as shown in Figure A-19.

```
* Pafish (Paranoid Fish) *

[-] Windows version: 10.0 build 18363
[-] Running in WoW64: True
[-] CPU: GenuineIntel
    Hypervisor: VBoxVBoxVBox
    CPU brand: 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using BeingDebugged via PEB access ... OK

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM exit ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... traced!
[*] Checking cpuid hypervisor vendor for known VM vendors ... traced!

[-] Generic reverse turing tests
[*] Checking mouse presence ... OK
[*] Checking mouse movement ... OK
[*] Checking mouse speed ... OK
[*] Checking mouse click activity ... OK
[*] Checking mouse double click activity ... OK
[*] Checking dialog confirmation ... traced!
[*] Checking plausible dialog confirmation ... traced!

[-] Generic sandbox detection
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... traced!
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via PEB access ... OK
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... OK
[*] Checking if pysical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... traced!
[*] Checking if operating system IsNativeVhdBoot() ... OK
```

Figure A-19: Pafish running in a VirtualBox VM

You can see here that Pafish detected my VM using several different indicators (denoted by the "traced!" message), such as CPU timing counters, lack of free disk space, and operating system uptime. Two tools similar to Pafish are Al-Khaser (*https://github.com/LordNoteworthy/al-khaser*) and InviZzzible (*https://github.com/CheckPointSW/InviZzzible*). Running multiple assessment tools inside your analysis VMs, both before and after you follow the guidance in this chapter, will give you an idea of how detectable the VMs are.

It's very difficult to completely conceal a VM from all the techniques that stress-testing software like Pafish and Al-Khaser use. After all, these tools were designed specifically for VM detection. Keep in mind that the goal of malware analysis isn't passing a stress test, and it's very unlikely that a malware sample in the wild would use all of these techniques.

That said, you can score higher on a stress test (and, of course, thwart malware) by using a bare-metal analysis system or instrumentation tools, both of which we'll touch on briefly at the end of this chapter.

Tips for Operational Security and Effectiveness

Operational security (OPSEC) is critical for malware analysis. Proper OPSEC includes safely handling both the malware and the investigation tools to protect yourself and others, including your organization if you analyze malware professionally.

Analyzing malware in any capacity is inherently risky. You could expose credentials or sensitive files from your host machine, especially if folder- and clipboard-sharing functionalities are enabled in your VM. You may unintentionally leak your home IP address to threat actors when investigating malicious infrastructure. Or, by allowing a malware sample to connect to a C2 server from your VM, you may tip off a threat actor to your investigation, which could have negative consequences. To mitigate these risks, this section contains some general tips for analyzing malware in your lab both safely and effectively.

Simulating Network Services

Detonating malware in a VM connected to the internet or even a local network carries risk, so a safer alternative is to simulate network services. Using tools such as INetSim and FakeDNS, you can trick the malware into believing it's operating in a networked or internet-accessible environment. INetSim can simulate many types of network services, such as FTP and HTTP, and FakeDNS specializes in simulating DNS services.

Network simulation is a simple process using the Remnux VM you set up earlier. First, make sure that both the Remnux and Windows VM network adapters are in Host-Only mode and that the Remnux VM is powered on. Issue the following command in a terminal in Remnux:

```
> accept-all-ips start
```

The *accept-all-ips* script configures the gateway (Remnux, in this case) to accept all IPv4 and IPv6 addresses and redirect them to the corresponding local port. Simply put, this enables Remnux to intercept, monitor, or manipulate network traffic destined to a certain IP address from the Windows VM.

Next, issue this command to start the INetSim service:

```
> inetsim
```

You should see output similar to that shown here:

```
remnux@remnux:~$ inetsim
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenberg
Using log directory:      /var/log/inetsim/
Using data directory:     /var/lib/inetsim/
Using report directory:   /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing configuration file.
Configuration file parsed successfully.
=== INetSim main process started (PID 1511) ===
Session ID:     1511
Listening on:   192.168.56.102
Real Date/Time: 2024-03-25 15:39:28
Fake Date/Time: 2024-03-25 15:39:28 (Delta: 0 seconds)
  Forking services...
    * smtps_465_tcp - started (PID 1518)
    * ftp_21_tcp - started (PID 1521)
    * smtp_25_tcp - started (PID 1517)
    * http_80_tcp - started (PID 1515)
    * pop3_110_tcp - started (PID 1519)
    * ftps_990_tcp - started (PID 1522)
    * pop3s_995_tcp - started (PID 1520)
    * https_443_tcp - started (PID 1516)
  done.
Simulation running.
--snip--
```

Then issue the `fakedns` command, like so:

```
> fakedns
```

This should generate output similar to the code shown here (you may not have as much output if your Windows VM is not yet powered on and communicating with the Remnux VM):

```
remnux@remnux:~$ fakedns
fakedns[INFO]: dom.query. 60 IN A 192.168.56.102
fakedns[INFO]: Response: au.download.windowsupdate.com -> 192.168.56.102
fakedns[INFO]: Response: api.msn.com -> 192.168.56.102
fakedns[INFO]: Response: slscr.update.microsoft.com -> 192.168.56.102
--snip--
```

Next, power up your Windows VM. After Windows is booted, test out FakeDNS and INetSim by navigating to your favorite website in a browser. If you've configured everything correctly, you should see something like the page shown in Figure A-20.

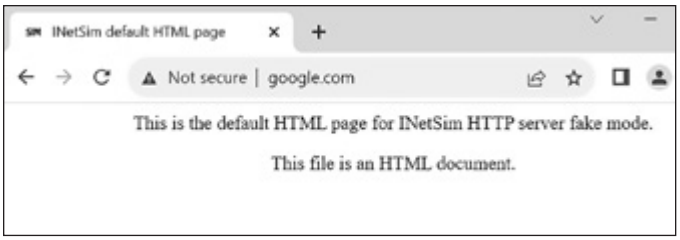


Figure A-20: INetSim and FakeDNS working correctly

INetSim and FakeDNS are successfully intercepting your web requests. Now, when you detonate malware in your Windows VM, the network connections will also be captured and can be analyzed later. When capturing and analyzing network traffic originating from your infected Windows VM, remember that Windows is quite noisy. Much of this traffic will be benign, so it's your job to filter out what's really of interest.

NOTE *INetSim stores detailed logs of network connections in* /var/log/inetsim*, and its configuration is stored in* /etc/inetsim/inetsim.conf. *Configuring INetSim is outside the scope of this chapter, but you can read more about it at* https://www.inetsim.org/documentation.html.

In addition to INetSim and FakeDNS, Wireshark and FakeNet are tools at your disposal for monitoring network traffic and capturing malicious activity safely.

Concealing Your IP

If you decide to connect your VM to the internet (by configuring the VM's network adapter in NAT or Bridged mode), you should always route your traffic through a VPN or similar technology to protect yourself. An additional benefit of using a VPN is that, depending on the VPN service provider, you may be able to choose your *exit node* (the point at which traffic exits the network). Some malware (for example, the SocGholish family) is targeted to a specific region or country, so if you're using an internet-connected VM for analysis, configuring the VPN exit node to a country that the malware is targeting can be a good analysis tactic.

Shared Folders and File Transferring

Given the risks of using clipboard sharing and shared folders, ideally these functions should be switched off unless you explicitly require them. Shared folders in particular are quite convenient for transferring malware files and other files between your host system and your VMs. If you choose to keep shared folders disabled (or if you didn't install any VM tools), you can copy files to and from your host by configuring FTP software such as FileZilla (*https://filezilla-project.org*). Simply configure an FTP server on your Linux VM and an FTP client on your Windows VM, for example, and then transfer files between them.

Updating Software

Keep your hypervisor software up to date. Hypervisor software is a prime target for malware authors, and it's not uncommon for vulnerabilities to be discovered and reported for software like VMware Workstation and VirtualBox. For reference, at the time of this writing, a quick search on the vulnerability database site CVEdetails.com showed 171 known vulnerabilities for VMware Workstation and 326 known vulnerabilities for VirtualBox! These vulnerabilities are not all critical, of course, but they're a risk to keep in mind. They could be used to attack your host operating system if not properly patched. You should also keep your VM guest software, such as VMware Tools and VirtualBox Guest Additions, updated to the latest version.

Bare-Metal Analysis

It's very difficult, if not impossible, to make a VM appear identical to a real, physical machine when you're dealing with advanced malware. You might be able to fool checks like querying the registry or enumerating running processes, but advanced malware will likely use more sophisticated tactics, such as CPU timing checks, or perhaps even currently unknown techniques. You may be able to circumvent these types of checks using techniques such as manually patching problematic areas of code (which can be very time-consuming) or using binary instrumentation techniques (discussed in the next section), but sometimes the best and most efficient solution is bare-metal analysis.

The term *bare metal* refers to an operating system running directly on the underlying hardware rather than virtualized in a hypervisor. This could be as simple as a spare laptop you've got lying around or as complex as a server rack full of physical devices with freshly installed operating systems. Detonating and analyzing malware on a bare-metal system is as close as you can get to how malware will actually behave on a real victim host. The hypervisor artifacts mentioned in this chapter and in Part II should be nonexistent, and more advanced VM detection techniques (such as CPU timing analysis) won't be effective. Bare-metal systems are even more powerful with some basic malware analysis tools installed. Just as in a VM, you might want to install tools such as a disassembler, a debugger, and process and network monitors, for example. In fact, I install many of the same tools in my bare-metal analysis system as in my analysis VMs.

While the positives of bare-metal analysis usually outweigh the negatives, there are a few things to be aware of. First and foremost, its effectiveness depends on your objectives. Second, since there's no underlying hypervisor, like VirtualBox or VMware, you can't take snapshots of a clean system. In VirtualBox, for example, after detonating a malware sample, you can simply revert the VM to a pristine state, which is not so easy with a bare-metal analysis setup. There are also special tools, such as Deep Freeze, Microsoft Deployment Toolkit (MDT), FOG Project, Clonezilla, and Rollback Rx. These tools allow snapshot-like capabilities, but they introduce

some amount of overhead, and this type of malware analysis environment doesn't scale very well. Additionally, while bare-metal systems won't have the hypervisor-related artifacts that malware can detect (such as registry keys and driver files on the disk), they might have other analysis tools installed that give you away.

Binary Instrumentation and Emulation

There are two more tools you might want to add to your malware analysis toolbox: binary instrumentation and emulation. *Binary instrumentation* is a method of modifying, or instrumenting, binary data and code to achieve some end result. In the context of malware analysis, binary instrumentation can be used to modify code to streamline the analysis process; this in turn will allow you to circumvent anti-analysis techniques. There are two primary forms of binary instrumentation: *dynamic binary instrumentation (DBI)* and *static binary instrumentation (SBI)*. DBI patches a program's instructions during runtime, and SBI makes changes to code prior to execution.

Binary instrumentation, specifically DBI, can complement other analysis tools, like debuggers. Using DBI, a reverse engineer can dynamically modify assembly instructions, which can be especially useful for analyzing context-aware malware. For example, problematic VM detection instructions such as `cpuid` and `rdtsc` can be modified or removed on the fly while the malware is running. Additionally, DBI is useful for monitoring and modifying Windows API calls and automating certain malware analysis tasks.

Binary instrumentation is not a silver bullet, however. DBI can introduce a lot of performance overhead, which can be problematic during the analysis process; it can also introduce time delays that malware might detect.

Binary instrumentation is a complex topic, so we won't go into more depth here, but some of the available binary instrumentation frameworks are summarized here:

DynamoRIO

A tool for manipulating and transforming code at runtime, while the target malware is executing. See *https://dynamorio.org*.

FRIDA

A dynamic instrumentation toolkit based on Python and JavaScript. See *https://frida.re* and also the post "Malware Analysis with Dynamic Binary Instrumentation Frameworks" from the BlackBerry Research & Intelligence Team at *https://blogs.blackberry.com/en/2021/04/malware-analysis-with-dynamic-binary-instrumentation-frameworks*.

Intel Pin

A popular dynamic binary instrumentation framework that is used as the base framework for many other instrumentation projects. See "Pin—A Dynamic Binary Instrumentation Tool" in Intel's developer

resources at *https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html*.

Two instrumentation tools built on Intel PIN are tiny_tracer and BluePill. The tiny_tracer project (*https://github.com/hasherezade/tiny_tracer*) is a tool that allows for dynamic logging (tracing) and manipulation of malware's code. It has built-in capabilities to bypass problematic anti-analysis features in malware. BluePill (*https://github.com/season-lab/bluepill*) is an older prototype tool designed with anti-analysis circumvention in mind. These are good examples of what can be done with dynamic instrumentation.

As opposed to binary instrumentation, *emulation* runs malicious code in a virtual, or emulated, environment. Emulation was discussed in the context of anti-malware software in Chapter 13, and it works in much the same way for malware analysis. It's also not as resource intensive as a complete sandbox environment or VM. Emulation allows for great control over malware and, similar to binary instrumentation, enables you to automate many analysis tasks. Here are some emulation frameworks you may want to explore:

Qiling Framework

A lightweight, cross-platform emulator that supports multiple software architectures. It also has support for many operating systems, including Windows, macOS, and Linux. See *https://qiling.io*.

Speakeasy

A modular emulator designed with malware in mind. It can emulate both user and kernel-mode malware. See *https://github.com/mandiant/speakeasy*.

Unicorn

A lightweight, multiplatform emulator framework. Qiling and Speakeasy are based on Unicorn. See *https://www.unicorn-engine.org*.

Because of their ability to supplement and automate parts of the malware analysis process, binary instrumentation tools and emulators can be formidable additions to your analysis toolbox. If you want to delve deeper into these topics, *Practical Binary Analysis* by Dennis Andriesse (No Starch Press, 2018) contains a lot more information.

Summary

This appendix discussed some fundamental concepts of arguably the most important part of malware analysis: the lab environment. You learned about the basic setup of an analysis lab environment, important safety principles, and some tools and techniques for concealing your malware analysis VMs and lab components from malware.

Concealing and hardening your analysis VMs can be a very effective, time-saving technique that helps circumvent many of the common

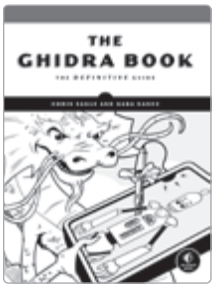
anti-analysis and VM detection tactics malware uses. However, there's one big downside to using these concealment techniques: you may be forgoing key intelligence about the malware's capabilities. If your goal is to truly understand a malware sample, concealing your VM can be counterproductive, as you could miss some of its most interesting evasion and detection behaviors.

RESOURCES

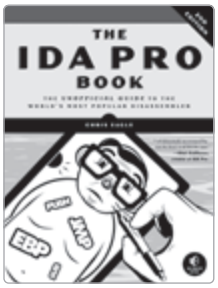
Visit *https://nostarch.com/evasive-malware* for errata and more information.

More no-nonsense books from **NO STARCH PRESS**

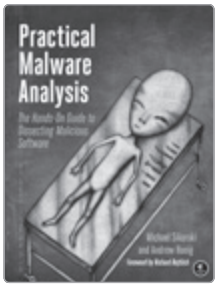
EVADING EDR
The Definitive Guide to Defeating Endpoint Detection Systems
BY MATT HAND
312 PP., $59.99
ISBN 978-1-7185-0334-2



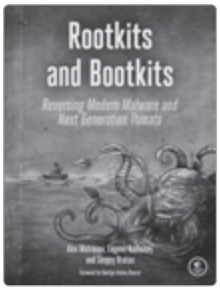
THE GHIDRA BOOK
The Definitive Guide
BY CHRIS EAGLE *AND* KARA NANCE
608 PP., $59.99
ISBN 978-1-7185-0102-7



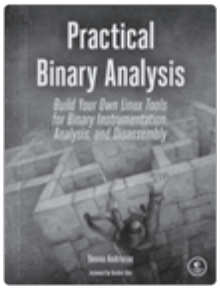
THE IDA PRO BOOK, 2ND EDITION
The Unofficial Guide to the World's Most Popular Disassembler
BY CHRIS EAGLE
672 PP., $79.99
ISBN 978-1-59327-289-0



PRACTICAL MALWARE ANALYSIS
The Hands-On Guide to Dissecting Malicious Software
BY MICHAEL SIKORSKI *AND* ANDREW HONIG
800 PP., $59.99
ISBN 978-1-59327-290-6



ROOTKITS AND BOOTKITS
Reversing Modern Malware and Next Generation Threats
BY ALEX MATROSOV, EUGENE RODIONOV, *AND* SERGEY BRATUS
448 PP., $49.95
ISBN 978-1-59327-716-1



PRACTICAL BINARY ANALYSIS
Build Your Own Linux Tools for Binary Instrumentation, Analysis, and Disassembly
BY DENNIS ANDRIESSE
456 PP., $59.99
ISBN 978-1-59327-912-7

PHONE:
800.420.7240 OR
415.863.9900

EMAIL:
SALES@NOSTARCH.COM

WEB:
WWW.NOSTARCH.COM

"A must-have for anyone who needs to identify, investigate, and analyze modern malware."

—ANUJ SONI, MALWARE REVERSE ENGINEER

We're all aware of Stuxnet, ShadowHammer, Sunburst, and similar attacks that use evasion to remain hidden while defending themselves from detection and analysis. Because advanced threats like these can adapt and, in some cases, self-destruct to evade detection, even the most seasoned investigators can use a little help with analysis now and then. *Evasive Malware* will introduce you to the evasion techniques used by today's malicious software and show you how to defeat them.

Following a crash course on using static and dynamic code analysis to uncover malware's true intentions, you'll learn how malware weaponizes context awareness to detect and skirt virtual machines and sandboxes, plus the various tricks it uses to thwart analysis tools. You'll explore the world of anti-reversing, from anti-disassembly methods and debugging interference to covert code execution and misdirection tactics. You'll also delve into defense evasion, from process injection and rootkits to fileless malware. Finally, you'll dissect encoding, encryption, and the complexities of malware obfuscators and packers to uncover the evil within.

You'll learn how malware:

- Abuses legitimate components of Windows, like the Windows API and LOLBins, to run undetected
- Uses environmental quirks and context awareness, like CPU timing and hypervisor enumeration, to detect attempts at analysis
- Bypasses network and endpoint defenses using passive circumvention techniques, like obfuscation and mutation, and active techniques, like unhooking and tampering
- Detects debuggers and circumvents dynamic and static code analysis

You'll also find tips for building a malware analysis lab and tuning it to better counter anti-analysis techniques in malware.

Whether you're a frontline defender, a forensic analyst, a detection engineer, or a researcher, *Evasive Malware* will arm you with the knowledge and skills you need to outmaneuver the stealthiest of today's cyber adversaries.

About the Author

KYLE CUCCI has over 17 years in cybersecurity and IT, including roles as a malware analyst and detection engineer with Proofpoint's Threat Research team and leader of the forensic investigations and malware research teams at Deutsche Bank. Cucci regularly speaks at security conferences and has led international trainings and workshops on topics such as malware analysis and security engineering. In his free time, Cucci enjoys contributing to the community via open source tooling, research, and blogging.

no starch press®

THE FINEST IN GEEK ENTERTAINMENT™

nostarch.com