

Malware Analysis Techniques

Tricks for the triage of adversarial software

Dylan Barker



Malware Analysis Techniques

Tricks for the triage of adversarial software

Dylan Barker



BIRMINGHAM—MUMBAI

Malware Analysis Techniques

Copyright © 2021 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Group Product Manager: Wilson Dsouza
Publishing Product Manager: Rahul Nair
Senior Editor: Arun Nadar
Content Development Editor: Sayali Pingale
Technical Editor: Sarvesh Jaywant
Copy Editor: Safis Editing
Project Coordinator: Shagun Saini
Proofreader: Safis Editing
Indexer: Pratik Shirodkar
Production Designer: Aparna Bhagat

First published: May 2021

Production reference: 1200521

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

978-1-83921-227-7

www.packt.com

Contributors

About the author

Dylan Barker is a technology professional with 10 years' experience in the information security space, in industries ranging from K12 and telecom to financial services. He has held many distinct roles, from security infrastructure engineering to vulnerability management. In the past, he has spoken at BSides events and has written articles for CrowdStrike, where he is currently employed as a senior analyst.

2

Static Analysis - Techniques and Tooling

Malware analysis is divided into two primary techniques: dynamic analysis, in which the malware is actually executed and observed on the system, and static analysis. Static analysis covers everything that can be gleaned from a sample without actually loading the program into executable memory space and observing its behavior.

Much like shaking a gift box to ascertain what we might expect when we open it, static analysis allows us to obtain a lot of information that may later provide context for behaviors we see in dynamic analysis, as well as static information that may later be weaponized against the malware.

In this chapter, we'll review several tools suited to this purpose, and several basic techniques for shaking the box that provide the best information possible. In addition, we'll take a look at two real-world examples of malware, and apply what we've learned to show how these skills and tools can be utilized practically to both understand and defeat adversarial software.

In this chapter, we will cover the following topics:

- The basics – hashing
- Avoiding rediscovery of the wheel
- Getting fuzzy
- Picking up the pieces

Technical requirements

The technical requirements for this chapter are as follows:

- FLARE VM set up, which we covered in the previous chapter
- An internet connection
- .zip files containing tools and malware samples from <https://github.com/PacktPublishing/Malware-Analysis-Techniques>

The basics – hashing

One of the most useful techniques an analyst has at their disposal is hashing. A **hashing algorithm** is a one-way function that generates a unique checksum for every file, much like a fingerprint of the file.

That is to say, every unique file passed through the algorithm will have a unique hash, even if only a single bit differs between two files. For instance, in the previous chapter, we utilized SHA256 hashing to verify whether a file that was downloaded from VirtualBox was legitimate.

Hashing algorithms

SHA256 is not the only hashing algorithm you're likely to come across as an analyst, though it is currently the most reliable in terms of balance of lack of collision and computational demand. The following table outlines hashing algorithms and their corresponding bits:

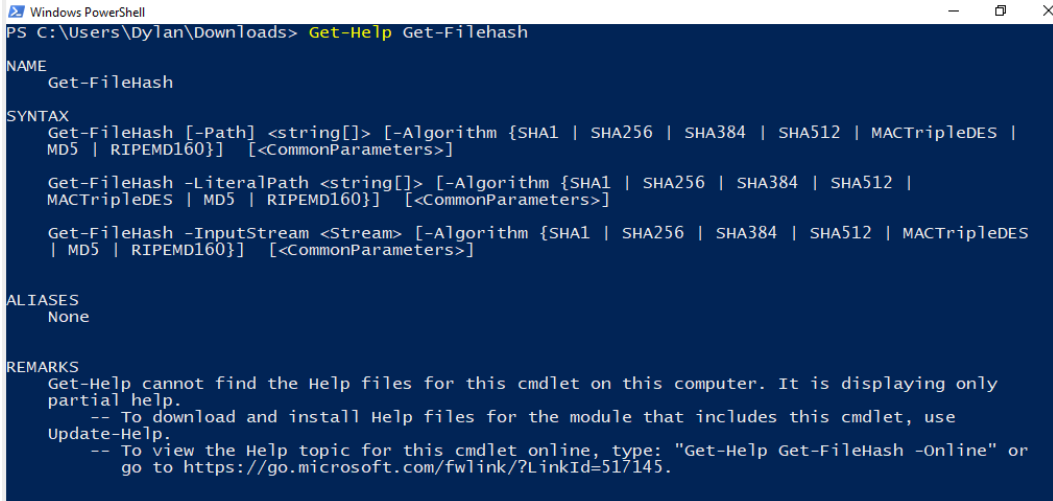
Algorithm	Output Bits	Broken
MD5	128	Yes
SHA1	160	Yes
SHA256	256	No
SHA512	512	No

Analysis Tip

In terms of hashing, **collision** is an occurrence where two different files have identical hashes. When a collision occurs, a hashing algorithm is considered broken and no longer reliable. Examples of such algorithms include MD5 and SHA1.

Obtaining file hashes

There are many different tools that can be utilized to obtain hashes of files within FLARE VM, but the simplest, and often most useful, is built into Windows PowerShell. `Get-FileHash` is a command we can utilize that does exactly what it says—gets the hash of the file it is provided. We can view the usage of the cmdlet by typing `Get-Help Get-FileHash`, as shown in the following screenshot:



```
Windows PowerShell
PS C:\Users\Dylan\Downloads> Get-Help Get-FileHash

NAME
    Get-FileHash

SYNTAX
    Get-FileHash [-Path] <string[]> [-Algorithm {SHA1 | SHA256 | SHA384 | SHA512 | MACTripleDES | MD5 | RIPEMD160}] [<CommonParameters>]

    Get-FileHash -LiteralPath <string[]> [-Algorithm {SHA1 | SHA256 | SHA384 | SHA512 | MACTripleDES | MD5 | RIPEMD160}] [<CommonParameters>]

    Get-FileHash -InputStream <Stream> [-Algorithm {SHA1 | SHA256 | SHA384 | SHA512 | MACTripleDES | MD5 | RIPEMD160}] [<CommonParameters>]

ALIASES
    None

REMARKS
    Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
    -- To download and install Help files for the module that includes this cmdlet, use Update-Help.
    -- To view the Help topic for this cmdlet online, type: "Get-Help Get-FileHash -Online" or go to https://go.microsoft.com/fwlink/?LinkId=517145.
```

Figure 2.1 – Get-FileHash usage

Analysis Tip

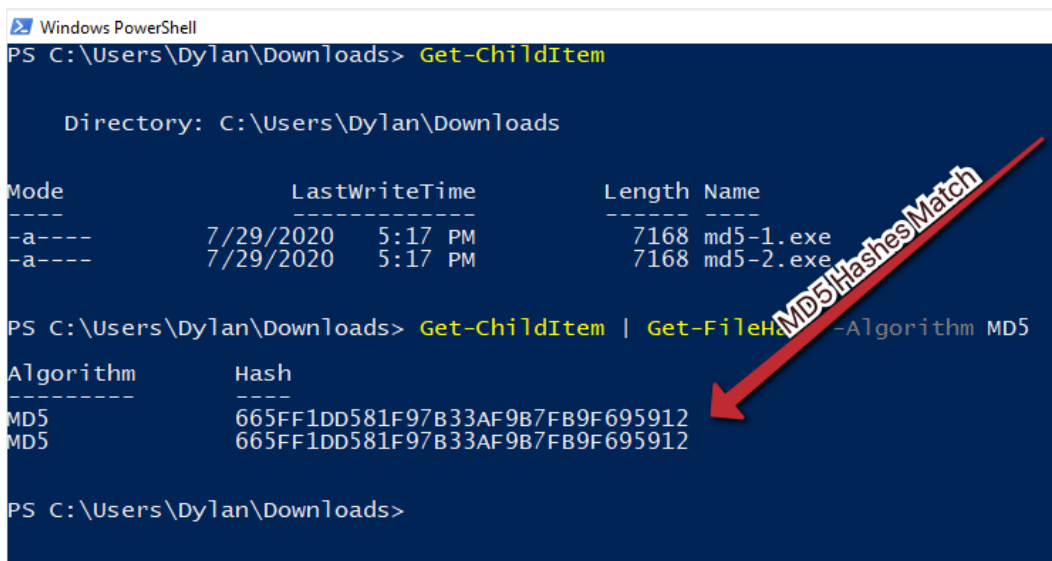
This section and many sections going forward will require you to transfer files from your host PC or download them directly to your analysis **virtual machine (VM)**. The simplest way to maintain isolation is to leave the network adapter on host-only and enable drag-and-drop or a shared clipboard via VirtualBox. Be sure to only do this on a clean machine, and disable it immediately when done via VirtualBox's **Devices** menu.

In this instance, there are two files available at <https://github.com/PacktPublishing/Malware-Analysis-Techniques>. These files are titled `md5-1.exe` and `md5-2.exe`. Once downloaded, `Get-FileHash` can be utilized on them, as shown in the next screenshot. In this instance, because there were the only two files in the directory, it was possible to use `Get-ChildItem` and pipe the output to `Get-FileHash`, as it accepts input from pipeline items.

Analysis Tip

Utilizing `Get-ChildItem` and piping the output to `Get-FileHash` is a great way to get the hashes of files in bulk and saves a great deal of time in triage, as opposed to manually providing each filename to `Get-FileHash` manually.

In the following screenshot, we can see that the files have the same MD5 hash! However, they also have the same size, so it's possible that these are, in fact, the same file:



```
Windows PowerShell
PS C:\Users\Dylan\Downloads> Get-ChildItem

Directory: C:\Users\Dylan\Downloads

Mode                LastWriteTime         Length Name
----                -
-a----           7/29/2020   5:17 PM             7168 md5-1.exe
-a----           7/29/2020   5:17 PM             7168 md5-2.exe

PS C:\Users\Dylan\Downloads> Get-ChildItem | Get-FileHash -Algorithm MD5

Algorithm      Hash
-----
MD5            665FF1DD581F97B33AF9B7FB9F695912
MD5            665FF1DD581F97B33AF9B7FB9F695912

PS C:\Users\Dylan\Downloads>
```

Figure 2.2 – The matching MD5 sums for our files

However, because MD5 is known to be broken, it may be best to utilize a different algorithm. Let's try again, this time with SHA256, as illustrated in the following screenshot:

```

Windows PowerShell
PS C:\Users\Dylan\Downloads> Get-ChildItem

Directory: C:\Users\Dylan\Downloads

Mode                LastWriteTime         Length Name
----                -
-a-----         7/29/2020   5:17 PM           7168 md5-1.exe
-a-----         7/29/2020   5:17 PM           7168 md5-2.exe

PS C:\Users\Dylan\Downloads> Get-ChildItem | Get-FileHash -Algorithm SHA256

Algorithm      Hash
-----
SHA256          E16A3E7BEA60AB2AA1E49E31199791648C58B14D1691935F25F3BD4E94F2F34B
SHA256          84AF18CFD067DF107B790EDDE3DBD23A0379F8FBBD1913AB0CEA74C4378F4569

Path
---
C:\Users\Dylan\Downloads\md5-1.exe
C:\Users\Dylan\Downloads\md5-2.exe

PS C:\Users\Dylan\Downloads>
  
```

SHA256 Hashes Differ!

Figure 2.3 – The SHA256 sums for our files

The SHA256 hashes differ! This indicates without a doubt that these files, while the same size and with the same MD5 hash, are *not* the same file, and demonstrates the importance of choosing a strong one-way hashing algorithm.

Avoiding rediscovery of the wheel

We have already established a great way of gaining information about a file via cryptographic hashing—akin to a file's fingerprint. Utilizing this information, we can leverage other analysts' hard work to ensure we do not dive deeper into analysis and waste time if someone has already analyzed our malware sample.

Leveraging VirusTotal

A wonderful tool that is widely utilized by analysts is VirusTotal. **VirusTotal** is a scanning engine that scans possible malware samples against several **antivirus (AV)** engines and reports their findings.

In addition to this functionality, it maintains a database that is free to search by hash. Navigating to <https://virustotal.com/> will present this screen:

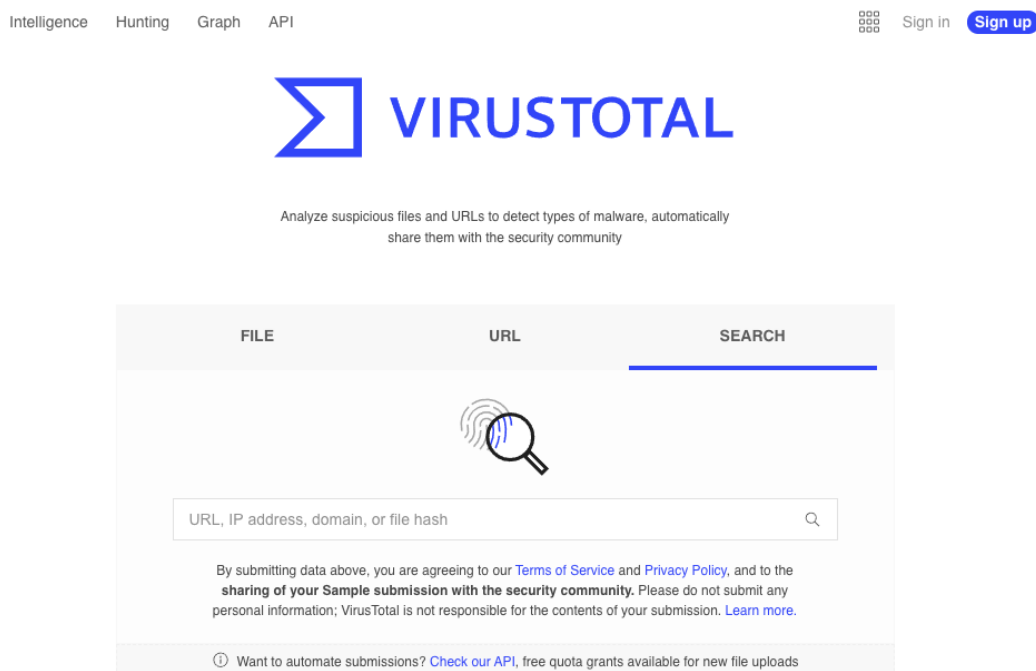


Figure 2.4 – The VirusTotal home page

In this instance, we'll use as an example a `275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabbf651fd0f` SHA256 hash. Entering this hash into VirusTotal and clicking the **Search** button will yield results as shown in the following screenshot, because several thousand analysts have submitted this file previously:

275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f

64 / 65

64 engines detected this file

275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f
eicar.com-1937

68.00 B
Size

2020-07-29 22:53:12 UTC
10 minutes ago

attachment text via-tor

DETECTION	DETAILS	RELATIONS	BEHAVIOR	COMMUNITY
Ad-Aware	① EICAR-Test-File (not A Virus)	AegisLab	① Test.File.EICAR.y	
AhnLab-V3	① Virus/EICAR_Test_File	Alibaba	① Trojan:MacOS/eicar.com	
ALYac	① Misc.Eicar-Test-File	Antiy-AVL	① TestFile/Win32.EICAR	
SecureAge APEX	① EICAR Anti-Virus Test File	Arcabit	① EICAR-Test-File (not A Virus)	
Avast	① EICAR Test-NOT Virus!!!	Avast-Mobile	① Eicar	
AVG	① EICAR Test-NOT Virus!!!	Avira (no cloud)	① Eicar-Test-Signature	
Baidu	① Win32.Test.Eicar.a	BitDefender	① EICAR-Test-File (not A Virus)	
BitDefenderTheta	① EICAR-Test-File (not A Virus)	Bkav	① DOS.EiracA.Trojan	
CAT-QuickHeal	① EICAR.TestFile	ClamAV	① Win.Test.EICAR_HDB-1	
CMC	① Eicar.test.file	Comodo	① ApplicUnwnt@#2975xIk8s2pq1	
Cynet	① Malicious (score: 85)	Cyren	① EICAR_Test_File	
DrWeb	① EICAR Test File (NOT A Virus!)	Elastic	① Eicar	
Emsisoft	① EICAR-Test-File (not A Virus) (B)	eScan	① EICAR-Test-File	

Figure 2.5 – VirusTotal search results for EICAR's test file

Within this screen, we can see that several AV engines correctly identify this SHA256 hash as being the hash for the **European Institute for Computer Antivirus Research (EICAR)** test file, a file commonly utilized to test the efficacy of AV and **endpoint detection and response (EDR)** solutions.

It should be apparent that utilizing our hashes first to search VirusTotal may greatly assist in reducing triage time and confirm suspected attribution much more quickly than our own analysis may.

However, this may not always be an ideal solution. Let's take a look at another sample—8888888.png. This file may be downloaded from <https://github.com/PacktPublishing/Malware-Analysis-Techniques>.

Warning!

888888.png is live malware—a sample of the **Qakbot (QBot)** banking Trojan threat! Handle this sample with care!

Utilizing the previous section's lesson, obtain a hash of the Qakbot file provided. Once done, paste the discovered hash into VirusTotal and click the search icon, as illustrated in the following screenshot:

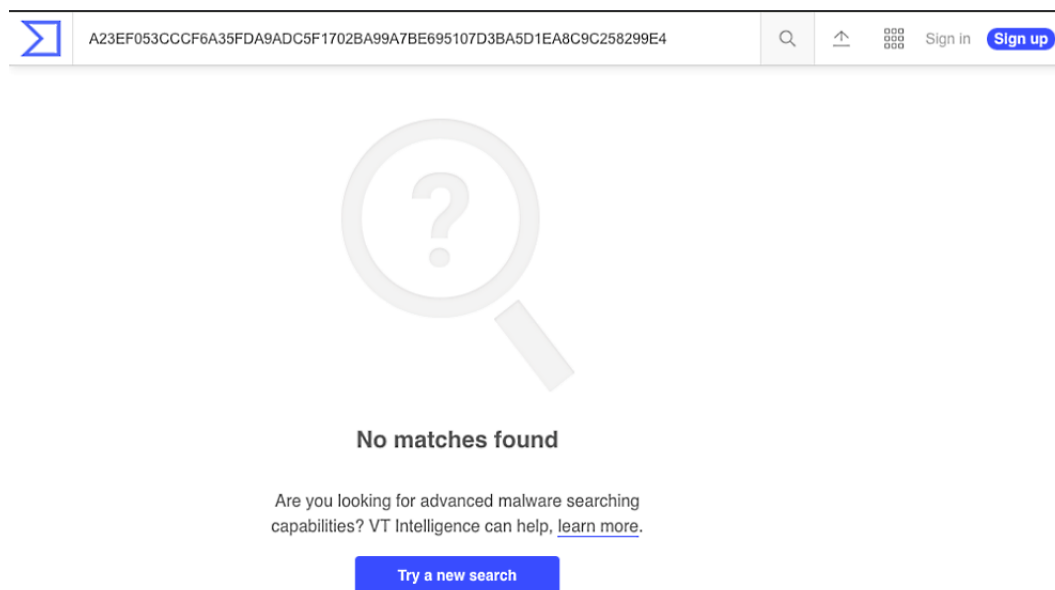


Figure 2.6 – Searching for the Qakbot hash yields no results!

It appears, based on the preceding screenshot, that this malware has an entirely unique hash. Unfortunately, it appears as though static cryptographic hashing algorithms will be of no use to our analysis and attribution of this file. This is becoming more common due to adversaries' implementation of a technique called **hashbusting**, which ensures each malware sample has a different static hash!

Analysis Tip

Hashbusting is quickly becoming a common technique among more advanced malware authors, such as the actor behind the EMOTET threat. Hashbusting implementations vary greatly, from adding in arbitrary snippets at compile-time to more advanced, probabilistic control flow obfuscation—such as the case with EMOTET.

Getting fuzzy

In the constant arms race of malware authoring and **Digital Forensics and Incident Response (DFIR)** analysts attempting to find solutions to common obfuscation techniques, hashbusting has also been addressed in the form of **fuzzy hashing**.

ssdeep is a fuzzy hashing algorithm that utilizes a similarity digest in order to create and output representations of files in the following format:

```
chunksize:chunk:double_chunk
```

While it is not necessary to understand the technical aspects of ssdeep for most analysts, a few key points should be understood that differentiate ssdeep and fuzzy hashing from standard cryptographic hashing methods such as MD5 and SHA256: *changing small portions of a file will not significantly change the ssdeep hash of the file, whereas changing one bit will entirely change the cryptographic hash.*

With this in mind, let's take a ssdeep hash of our 8888888.png sample. Unfortunately, ssdeep is not installed by default in FLARE VM, so we will require a secondary package. This can be downloaded from <https://github.com/PacktPublishing/Malware-Analysis-Techniques>. Once the ssdeep binaries have been extracted to a folder, place the malware sample in the same folder, as shown in the following screenshot:

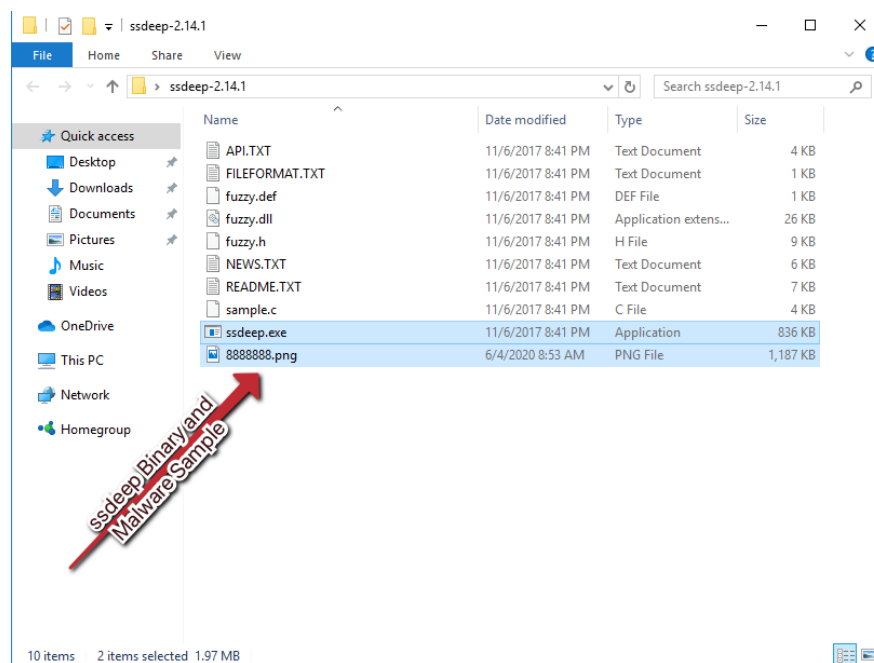


Figure 2.7 – Place the binary into the same folder as your ssdeep executable for ease of use

Next, we'll need to open a PowerShell window to this path. There's a quick way to do this in Windows—click in the path bar of Explorer, type `powershell.exe`, strike *Enter*, and Windows will helpfully open a PowerShell prompt at the current path! This is illustrated in the following screenshot:

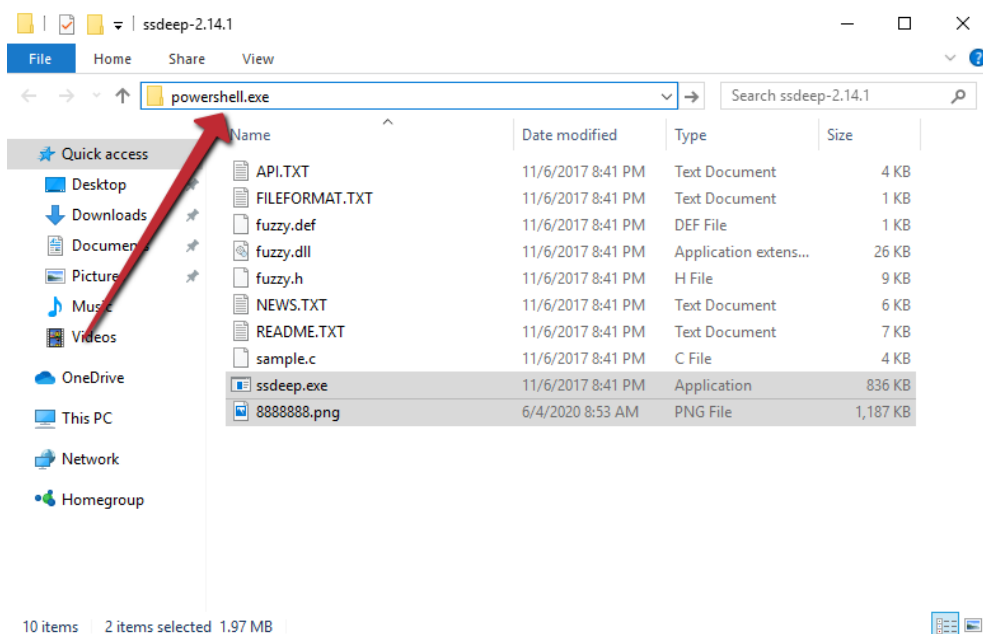


Figure 2.8 – An easy shortcut to open a PowerShell prompt at the current folder's pathing

With PowerShell open at the current prompt, we can now utilize the following to obtain our ssdeep hash: `.\ssdeep.exe .\8888888.png`. This will then return the ssdeep fuzzy hash for our malware sample, as illustrated in the following screenshot:

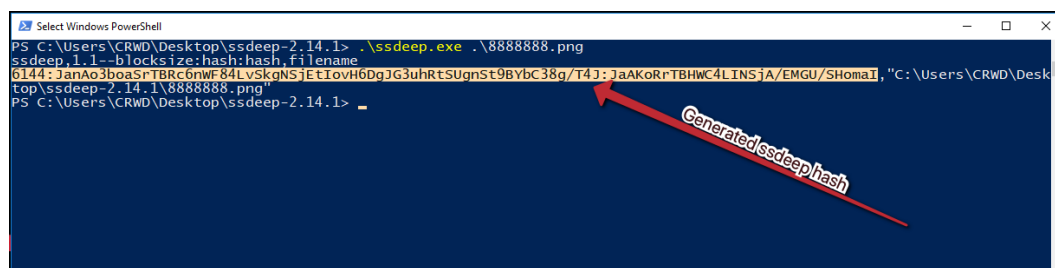


Figure 2.9 – The ssdeep hash for our Qbot sample

We can see that in this instance, the following fuzzy hash has been returned:

```
6144:JanAo3boaSrTBRC6nWF84LvSkgNSjEtIovH6DgJG3uhRtSUgnSt9BYbC  
38g/T4J:JaAKoRrTBHWC4LINSjA/EMGU/ShomaI
```

Unfortunately, at this time, the only reliable publicly available search engine for ssdeep hashes is VirusTotal, which requires an Enterprise membership. However, we'll walk through the process of searching VirusTotal for fuzzy hashes. In the VirusTotal Enterprise home page, ssdeep hashes can be searched with the following:

```
ssdeep:"<ssdeephashhere>"
```

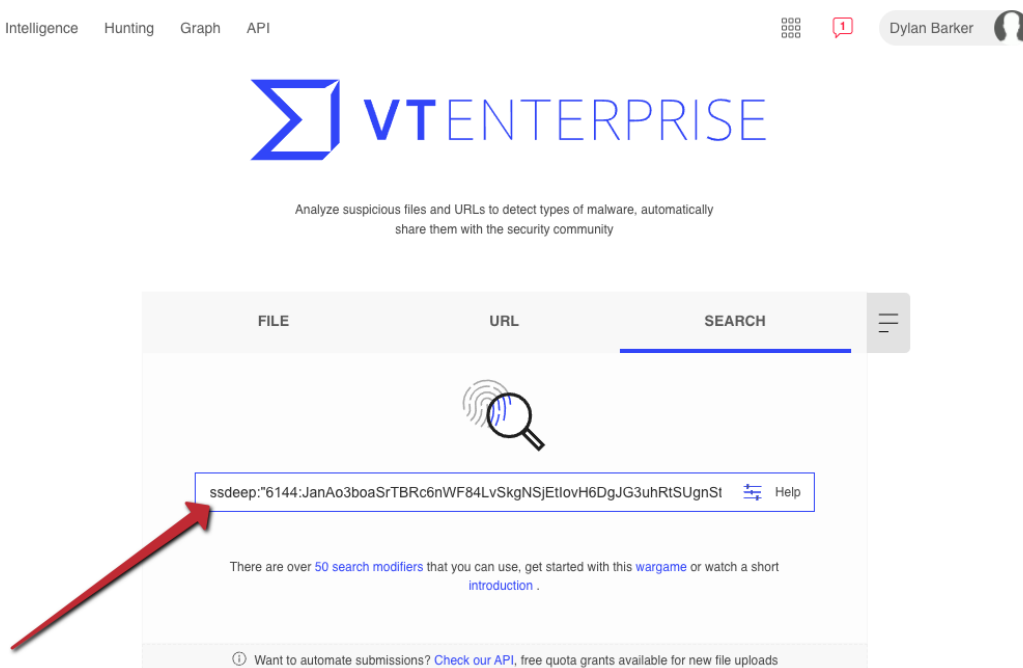


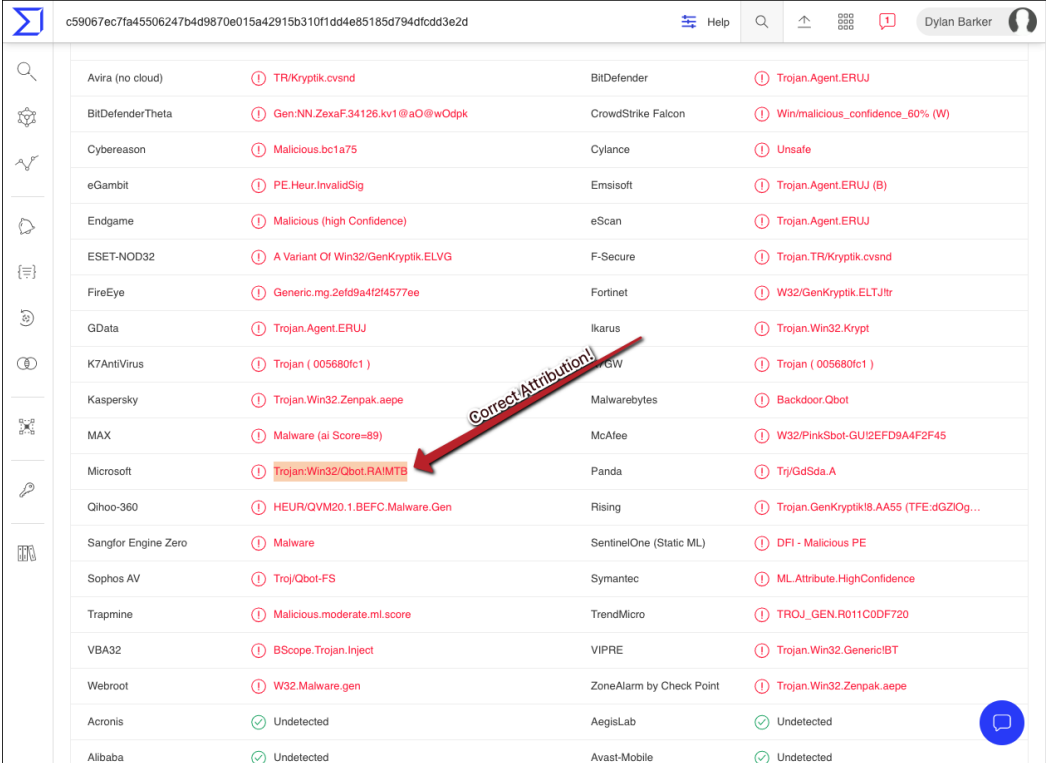
Figure 2.10 – ssdeep search syntax on VirusTotal

Because comparing fuzzy hashes requires more computational power than searching rows for fixed, matching cryptographic hashes, VirusTotal will take a few moments to load the results. However, once it does, you will be presented with the page shown in the following screenshot, containing a wealth of information, including a corresponding cryptographic hash, when the sample was seen, and engines detecting the file, which will assist with attribution:

Files	Similarity	Detections	First seen	Last seen	Submitters
C59867EC7FA455862478409878E815A42915B318F7D04E851850794DFC003E2D 2ef09a4f2f4577ee8846cb7175e873bb.virus	98%	44 / 72	2020-06-07 10:50:55	2020-06-07 10:50:55	1
2285F884C4530ED3E83A864FCC661811837844598AB46CB0ED68854325E657D 30ee3ea2707ae70a255d0a84008c47a.virus	98%	44 / 73	2020-06-07 10:52:24	2020-06-07 10:52:24	1
4058C28875E788A36D322E8AB01155F2097497818FC041F987033A95F0EE dad30a48a98afae2bfb355d65622cb0.virus	98%	41 / 73	2020-06-07 10:50:54	2020-06-07 10:50:54	1
2781C3CAE3F63213C90888EC3A6C5785614E2CC6F8C99598F9ACBE81E9CA53 539c9a2e554b12fe98cb17444044b43b.virus	98%	41 / 73	2020-06-07 10:50:42	2020-06-07 10:50:42	1
64CECC473F1E82F178F23EF08F702F50F51CC7E89828EF701870DC8418A23CF \\Users\Petra\AppData\Roaming\Microsoft\Azuleq\uuuuy.exe	98%	50 / 73	2020-06-12 00:16:06	2020-06-12 00:16:06	1
A6C727274CD43B46589768428A886F5624A2B6432EA58A7ED1C8AC8C5FA79E \\Users\Petra\AppData\Roaming\Microsoft\Azuleq\uuuuy.exe	92%	42 / 73	2020-06-07 20:45:26	2020-06-07 20:45:26	1
287F8CB45AC21628A1E1AA2E97DC79662235984E5795A6158CB823C9A508F83D %APPDATA%\microsoft\velea\vuauu.exe	92%	21 / 73	2020-06-04 18:03:05	2020-06-04 18:03:05	1
795E155672C21662E7B23303314E66F91E638831C716E2E167EBAB94927BF9A %APPDATA%\microsoft\yxtwvbw\ynmel.exe	92%	41 / 70	2020-06-08 09:46:48	2020-06-08 09:46:48	1

Figure 2.11 – Fuzzy hash search results for our Qbot sample on VirusTotal

Clicking one of the highly similar cryptographic hashes will load the VirusTotal scan results for the sample and show what our sample likely is, as illustrated in the following screenshot:



Vendor	Detection	Vendor	Detection
Avira (no cloud)	① TR/Kryptik.cvnsd	BitDefender	① Trojan.Agent.ERUJ
BitDefenderTheta	① Gen:NN.ZexaF.34126.kv1@aO@wOdpk	CrowdStrike Falcon	① Win/malicious_confidence_60% (W)
Cybereason	① Malicious.bc1a75	Cylance	① Unsafe
eGambit	① PE.Heur.InvalidSig	Emsisoft	① Trojan.Agent.ERUJ (B)
Endgame	① Malicious (high Confidence)	eScan	① Trojan.Agent.ERUJ
ESET-NOD32	① A Variant Of Win32/GenKryptik.ELVG	F-Secure	① Trojan.TR/Kryptik.cvnsd
FireEye	① Generic.mg.2efd9a4f24577ee	Fortinet	① W32/GenKryptik.ELTJlr
GData	① Trojan.Agent.ERUJ	Ikarus	① Trojan.Win32.Krypt
K7AntiVirus	① Trojan (005680fc1)	Malwarebytes	① Trojan (005680fc1)
Kaspersky	① Trojan.Win32.Zenpak.aepe	McAfee	① W32/PinkBot-GU!2EFD9A4F2F45
MAX	① Malware (ai Score=89)	Panda	① Trj/GdSda.A
Microsoft	① Trojan:Win32/Obot.RA!MTB	Rising	① Trojan.GenKryptik!8.AA55 (TFE:dGZIOg...
Qihoo-360	① HEUR/QVM20.1.BEFC.Malware.Gen	SentinelOne (Static ML)	① DFI - Malicious PE
Sangfor Engine Zero	① Malware	Symantec	① ML.Attribute.HighConfidence
Sophos AV	① Troj/Qbot-FS	TrendMicro	① TROJ_GEN.R011C0DF720
Trapmine	① Malicious.moderate.ml.score	VIPRE	① Trojan.Win32.Generic!BT
VBA32	① BScope.Trojan.Inject	ZoneAlarm by Check Point	① Trojan.Win32.Zenpak.aepe
Webroot	① W32.Malware.gen		
Acronis	✓ Undetected	AegisLab	✓ Undetected
Alibaba	✓ Undetected	Avast-Mobile	✓ Undetected

Figure 2.12 – Scan results of highly similar files that have been submitted to VirusTotal

If you do not have a VirusTotal Enterprise subscription, all is not lost in terms of fuzzy hashing, however. It is possible to build your own database or compare known samples of malware to the fuzzy hashes of new samples. For full usage of `ssdeep`, see their project page at <https://ssdeep-project.github.io/ssdeep/usage.html>.

Picking up the pieces

In addition to simple fingerprints of files, be they fuzzy or otherwise, a file can give us several other basic pieces of information about it without executing. Attackers have a few simple tricks that are frequently used to attempt to slow down analysis of malware.

Malware serotyping

Take, for instance, our current sample—8888888.png; if we open this file as a .png image, it appears to be corrupt!

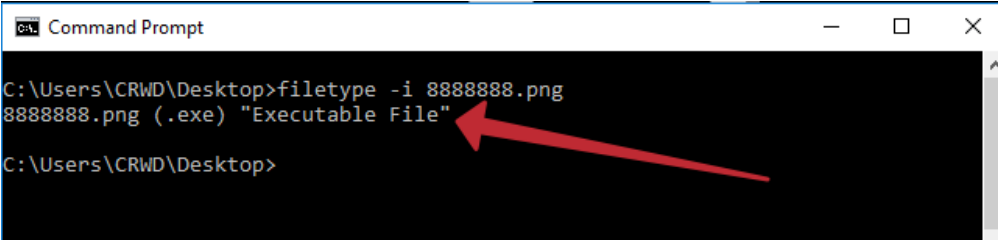
Adversaries frequently change the extension of files, sometimes excluding it altogether and sometimes creating double extensions, such as notmalware.doc.exe, in order to attempt to obfuscate their intentions, bypass EDR solutions, or utilize social engineering to entice a user into executing their payload.

Fortunately for malware analysts, changing a file's extension does not hide its true contents, and serves only as an aesthetic change in most regards. In computing, all files have a header that indicates to the operating system how to interpret the file. This header can be utilized to *type* a file, much like a crime forensic analyst would type a blood sample. See the following table for a list of common file headers related to malware:

Header	File Type
MZ	Windows PE (.exe, .dll)
PK..	ZIP file formats (.zip, .docx, .apk, .jar)
Rar!....	WinRAR archives
.ELF	Linux ELF executable
X.S.BB`	Mac disk image file
%PDF-	PDF document
MSCF	Microsoft cabinet files (.cab)

Unix and Unix-like systems have a built-in utility for testing file types, called `file`. Unfortunately, Windows lacks this ability by default, and requires a secondary tool installation within FLARE. `filetype.exe` is a good choice for this and can be obtained from <https://github.com/PacktPublishing/Malware-Analysis-Techniques>.

Once extracted, we can use `filetype.exe -i 8888888.png` to ascertain what the file really is. In this case, `filetype` returns that this is a Windows PE file, as illustrated in the following screenshot:



```

C:\Users\CRWD\Desktop>filetype -i 8888888.png
8888888.png (.exe) "Executable File"
C:\Users\CRWD\Desktop>
  
```

A red arrow points to the output line: `8888888.png (.exe) "Executable File"`.

Figure 2.13 – Results from utilizing `filetype.exe`; our image is actually a Windows Portable Executable!

Analysis Tip

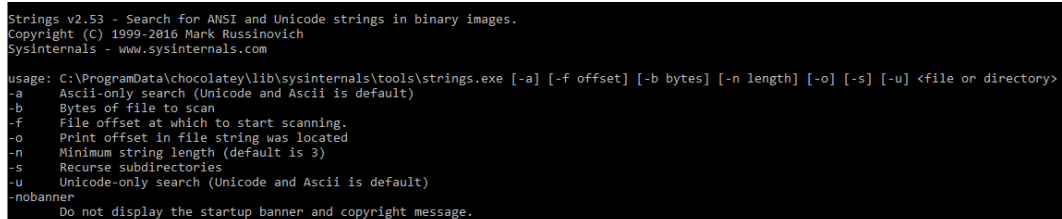
While tools exist to automatically ascertain the file type, such as Unix's `FILE` and `FILETYPE` for Windows, it's also possible to use a hexadecimal editor such as 010 Editor to simply examine the file's header and compare it to known samples.

Collecting strings

When an executable is compiled, certain ASCII- or Unicode-encoded strings used during development may be included in the binary.

The value of intelligence held by strings in an executable should not be underestimated. They can offer valuable insight into what a file may do upon execution, which command-and-control servers are being utilized, or even who wrote it.

Continuing with our sample of QBot, a tool from Microsoft's Windows Sysinternals can be utilized to extract any strings located within the binary. First, let's take a look at some of the command-line switches that may assist in making the Strings tool as useful as possible, as illustrated in the following screenshot:



```
Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

usage: C:\ProgramData\chocolatey\lib\sysinternals\tools\strings.exe [-a] [-f offset] [-b bytes] [-n length] [-o] [-s] [-u] <file or directory>
-a      Ascii-only search (Unicode and Ascii is default)
-b      Bytes of file to scan
-f      File offset at which to start scanning.
-o      Print offset in file string was located
-n      Minimum string length (default is 3)
-s      Recurse subdirectories
-u      Unicode-only search (Unicode and Ascii is default)
-nobanner
        Do not display the startup banner and copyright message.
```

Figure 2.14 – Command-line options for the Strings utility

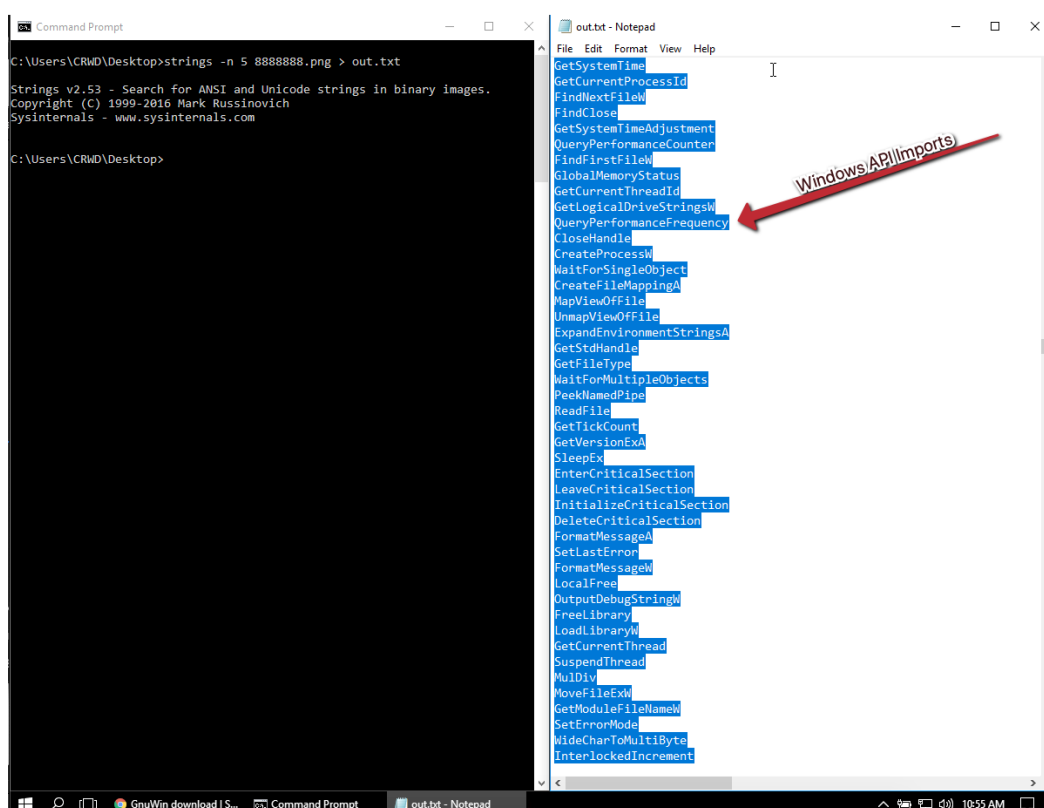
As shown, ASCII and Unicode strings are both searched by default—this is ideal, as we'd like to include both in our search results to ensure we have the most intelligence possible related to our binary. The primary switch we are concerned with is `-n`, the minimum string length to return. It's generally recommended to utilize a value of 5 for this switch, otherwise garbage output may be encountered that may frustrate analysis.

Let's examine which strings our Qbot sample contains, with `strings -n 5 88888888.png > output.txt`.

Analysis Tip

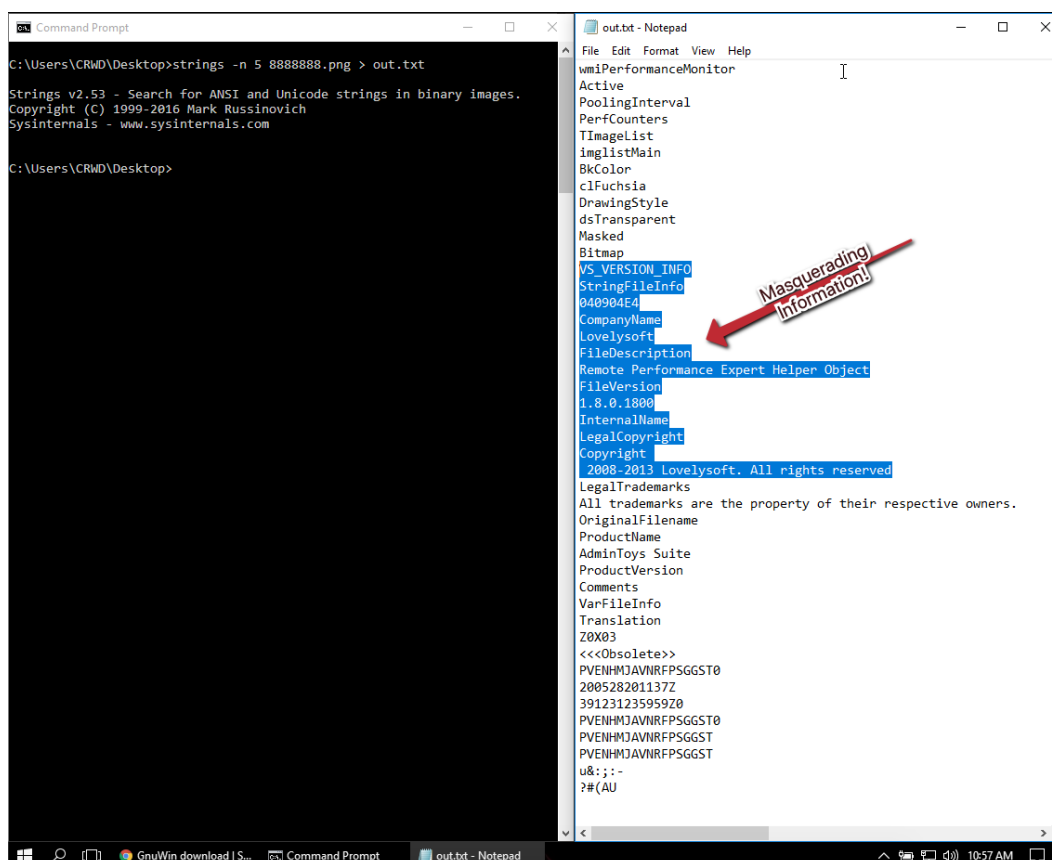
The > operator on the Windows command line will redirect the terminal's standard output to a file or location of your choosing, handy if you don't want to scroll through the terminal or truncate output. Similarly, >> will append standard output to the end of an already existing file.

Once this command is issued, a new text document will be created. Taking a look at our text file, we can see several strings have been returned, including some of the Windows **application programming interface (API)** modules that are imported by this binary—these may give a clue to some of the functionality the malware offers and are illustrated in the following screenshot:



Figures 2.15 – Output of strings showing modules imported from the Windows API, as well as information on which executable may have served as the basis of this payload

Scrolling down to the end of the output, we can gain some information on which executable was backdoored or what the binary is masquerading as! This may prove useful both in tracking the operations of the campaign and tracking **indicators of compromise (IOCs)** for internal outbreaks. The information can be seen in the following screenshot:



```

C:\Users\CRWD\Desktop>strings -n 5 8888888.png > out.txt

Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Users\CRWD\Desktop>

wmiperformanceMonitor
Active
PoolingInterval
PerfCounters
TImageList
imglistMain
BkColor
clFuchsia
DrawingStyle
dsTransparent
Masked
Bitmap
VS_VERSION_INFO
StringFileInfo
040904E4
CompanyName
Lovelysoft
FileDescription
Remote Performance Expert Helper Object
FileVersion
1.8.0.1800
InternalName
LegalCopyright
Copyright
2008-2013 Lovelysoft. All rights reserved
LegalTrademarks
All trademarks are the property of their respective owners.
OriginalFilename
ProductName
AdminToys Suite
ProductVersion
Comments
VarFileInfo
Translation
20X03
<<<Obsolete>>>
PVENH#MJAVNRFPSSGGST0
200528201137Z
391231235959Z0
PVENH#MJAVNRFPSSGGST0
PVENH#MJAVNRFPSSGGST
PVENH#MJAVNRFPSSGGST
u&;:-
?#(AU
  
```

Figures 2.16 – Output of strings showing modules imported from the Windows API, as well as information on which executable may have served as the basis of this payload

As you can see, information gained via this methodology may prove useful both in tracking the operations of the campaign and tracking IOCs for internal outbreaks.

Challenges

The malware samples for these challenges can be found at <https://github.com/PacktPublishing/Malware-Analysis-Techniques>.

Challenge 1

Attempt to answer the following questions utilizing what you've learned in this chapter—remembering that you are working with live malware. Do not execute the sample!

1. What is the SHA256 hash of the sample?
2. What is the ssdeep hash of the sample?
3. Can you attribute this sample to a particular malware family?

Challenge 2

In 2017, malware researcher Marcus Hutchins (@MalwareTechBlog) utilized the Strings utility to stop the global threat of WannaCry by identifying and sinkholing a kill-switch domain.

Utilizing the second sample, can you correctly identify the kill-switch domain?

Summary

In this chapter, we've taken a look at some basic static analysis techniques, including generating static file fingerprints using hashing, fuzzy hashing when this is not enough, utilizing **open source intelligence (OSINT)** such as VirusTotal to avoid replicating work, and understanding strings that are present within a binary after compilation.

While basic, these techniques are powerful and comprise a base skillset required to be effective as a malware analyst, and we will build on each of these techniques in the coming chapters to perform more advanced analysis. To test your knowledge of the chapter, make sure you have gone through the *Challenges* section and seen how your static analysis skills stack up against real-world adversaries. In the next chapter, we'll be moving on from basic static analysis to dynamic analysis—actually executing our malware!

Further reading

ssdeep advanced usage: <https://ssdeep-project.github.io/ssdeep/usage.html>