

Intro to Malware Analysis

Task 1 Introduction

Every once in a while, when you are working as a SOC analyst, you will come across content (a file or traffic) that seems suspicious, and you will have to decide whether that content is malicious or not. It is normal to feel confused with all the mixed signals that such content provides. This becomes a little overwhelming for somebody who is just starting in Cybersecurity, and it is common to begin self-guessing oneself. Knowing what steps to take to resolve such a scenario is helpful. This room will lay down some steps to help you make the initial conclusion about a particular suspicious file.

Notably, in this room, you will learn:

- What is malware?
- How to start analyzing a malware
- Static and Dynamic malware analysis
- Resources to help you analyze malware

So let's get started!

Answer the questions below

Let's learn Malware Analysis!

No answer needed

Task 2 Malware Analysis

Malware

The word malware is derived from the term MALicious softWARE. Therefore, any software that has a malicious purpose can be considered malware. Malware is further classified into different categories based on its behavior. However, we will not go into the details of those in this room. Here we will ponder the steps we will take if we suspect that we found malware in a machine. So, let's get started.

The purpose behind Malware Analysis

Malware Analysis is an important skill to have. As a quick overview, Malware Analysis is performed by the following people in the Security Industry:

- **Security Operations** teams analyze malware to write detections for malicious activity in their networks.
- **Incident Response** teams analyze malware to determine what damage has been done to an environment to remediate and revert that damage.
- **Threat Hunt** teams analyze malware to identify IOCs, which they use to hunt for malware in a network.
- **Malware Researchers** in security product vendor teams analyze malware to add detections for them in their security products.
- **Threat Research** teams in OS Vendors like Microsoft and Google analyze malware to discover the vulnerabilities exploited and add more security features to the OS/applications.

Overall, it seems like many different people do malware Analysis for many compelling reasons. So let's see how to start!

Before we begin!



Please note that malware is like a weapon because it can produce great harm if not handled with care. For this reason, always take the following precautions while analyzing malware:

- Never analyze malware or suspected malware on a machine that does not have the sole purpose of analyzing malware.

- When not analyzing or moving malware samples around to different locations, always keep them in password-protected zip/rar or other archives so that we can avoid accidental detonation.
- Only extract the malware from this password-protected archive inside the isolated environment, and only when analyzing it.
- Create an isolated VM specifically for malware analysis, which has the capability of being reverted to a clean slate once you are done.
- Ensure that all internet connections are closed or at least monitored.
- Once you are done with malware analysis, revert the VM to its clean slate for the next malware analysis session to avoid residue from a previous malware execution corrupting the next one.

Answer the questions below

Which team uses malware analysis to look for IOCs and hunt for malware in a network?

Threat hunt team

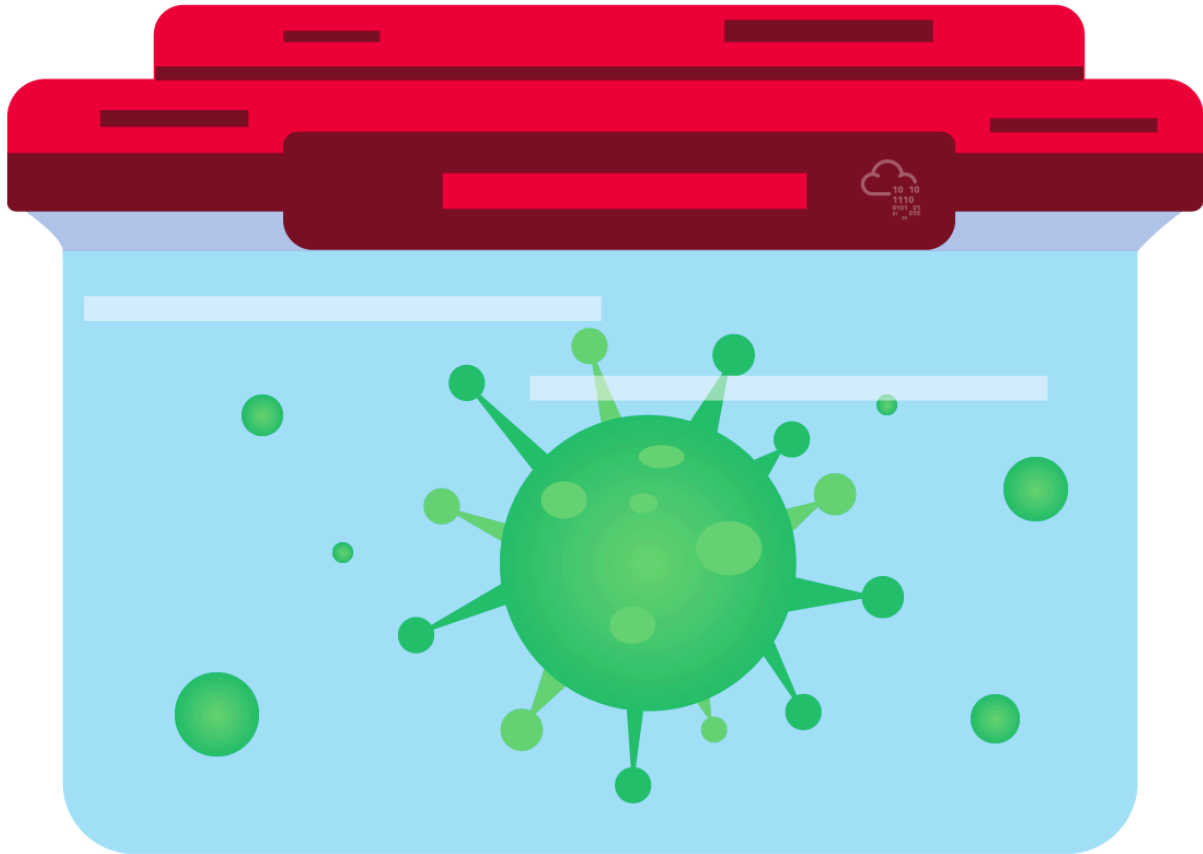
Task 3 Techniques of malware analysis

Malware Analysis is like solving a puzzle. Different tools and techniques are used to find the pieces of this puzzle, and joining those pieces gives us the complete picture of what the malware is trying to do. Most of the time, you will have an executable file (also called a binary or a PE file. PE stands for Portable Executable), a malicious document file, or a Network Packet Capture (Pcap). The Portable Executable is the most prevalent type of file analyzed while performing Malware Analysis.

To find the different puzzle pieces, you will often use various tools, tricks, and shortcuts. These techniques can be grouped into the following two categories:

- Static Analysis
- Dynamic Analysis

Static Analysis

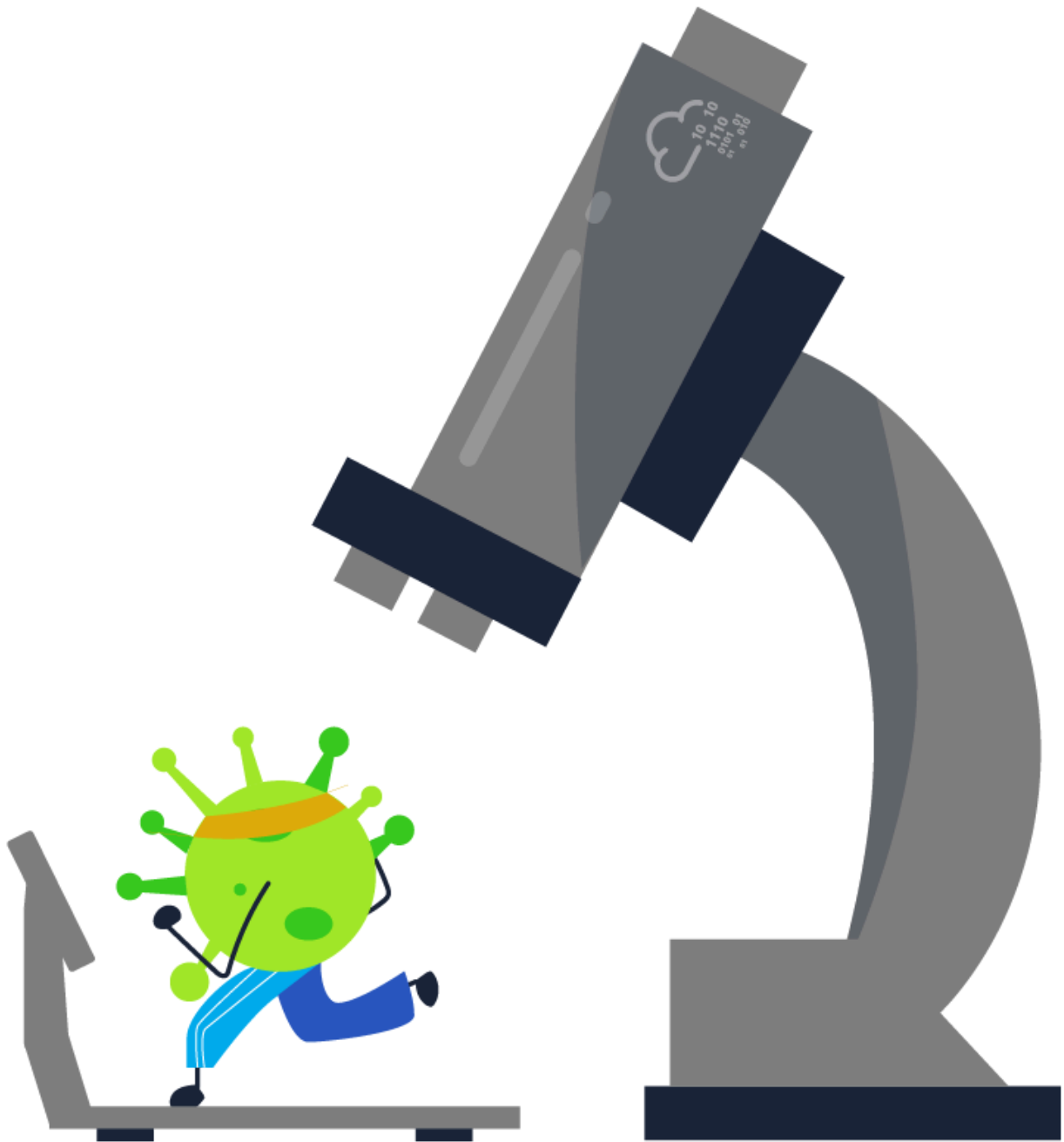


When malware is analyzed without being executed, it is called Static Analysis. In this case, the different properties of the PE file are analyzed without running it. Similarly, in the case of a malicious document, exploring the document's properties without analyzing it will be considered Static Analysis. Examples of static analysis include checking for strings in malware, checking the PE header for information related to different sections, or looking at the code using a disassemble. We will look at some of these techniques later in the room.

Malware often uses techniques to avoid static analysis. Some of these techniques use obfuscation, packing, or other means of hiding its properties. To circumvent these techniques, we often use dynamic analysis.

Dynamic Analysis

Malware faces a dilemma. It has to execute to fulfil its purpose, and no matter how much obfuscation is added to the code, it becomes an easy target for detection once it runs.



Static analysis might provide us with crucial information regarding malware, but sometimes that is not enough. We might need to run the malware in a controlled environment to observe what it does in these cases. Malware can often hide its properties to thwart Static Analysis. However, in most of those cases, Dynamic Analysis can prove fruitful. Dynamic analysis techniques include running the malware in a VM, either in a manual fashion with tools installed to monitor the malware's activity or in the form of sandboxes that perform this task automatically. We will learn

about some of these techniques later in this room. Once we run the malware in a controlled environment, we can use our knowledge from the Windows Forensics rooms to identify what it did in our environment. The advantage here is that since we control the environment, we can configure it to avoid noise, like activity from a legitimate user or Windows Services. Thus, everything we observe in such an environment points to malware activity, making it easier to identify what the malware did in this scenario.

Malware, however, often uses techniques to prevent an analyst from performing dynamic analysis. Since most dynamic analysis is performed in a controlled environment, most methods to bypass dynamic analysis include detecting the environment in which it is being run. Therefore, in these cases, the malware uses a different, benign code path if it identifies that it is being run in a controlled environment.

Advanced Malware Analysis

Advanced malware analysis techniques are used to analyze malware that evades basic static and dynamic analysis. For performing advanced malware analysis, disassemblers and debuggers are used. Disassemblers convert the malware's code from binary to assembly so that an analyst can look at the instructions of the malware statically. Debuggers attach to a program and allow the analyst to monitor the instructions in malware while it is running. A debugger allows the analyst to stop and run the malware at different points to identify interesting pieces of information while also providing an overview of the memory and CPU of the system. We will not cover advanced malware analysis in this room. However, it will be covered in a future module targeting malware analysis.

Answer the questions below

Which technique is used for analyzing malware without executing it?

Static analysis

Which technique is used for analyzing malware by executing it and observing its behavior in a controlled environment?

Dynamic analysis

Task 4 Basic Static Analysis

When analyzing a new piece of malware, the first step is usually performing basic static analysis. Basic static analysis can be considered sizing up the malware, trying to find its properties before diving deep into analysis. It provides us with an overview of what we are dealing with. Sometimes it might give us some critical information, for example, what API calls the malware is making or whether it's packed or not. However, other times, it might only give us

information to help us size the malware up and give us an idea of the effort required to analyze it.

So without further ado, let's see some of the techniques we can use to perform basic static analysis.

Caution!

Although static analysis is performed without running the malware, it is highly recommended that you perform malware analysis in an isolated Virtual Machine. You can create a clean snapshot of your Virtual Machine before performing any malware analysis and revert it to start from a clean state again after every analysis. Don't perform malware analysis on a live machine not purpose-built for malware analysis. For this room, we will be using the attached Remnux VM. Remnux (Reverse Engineering Malware Linux) is a Linux distribution purpose-built for malware analysis. It has many tools required for malware analysis already installed on it.

Accessing the Attached VM:

For the following steps, we will use the attached VM. Start the machine by clicking on the Start Machine button in the top-right corner of this task.

The machine will start in the split view. Alternatively, you can access the machine using the following credentials:

Username: ubuntu

Password: 123456

Examining the file type

Though often the file type of malware is visible in the file extension and is obvious, sometimes malware authors try to trick users by using misleading file extensions. In such scenarios, it is helpful to know how to find the actual file type of a file without depending on file extensions. In Linux, we can find the file type of a file using the file command. To understand what the file command does, we can read its man page or use the --help option:

```
man file or file --help
```

We will find out that it is a simple command to use. We can use the following command to find the file type of a file:

```
file <filename>  
Remnux
```

```
user@machine$ file wannacry
wannacry: PE32 executable (GUI) Intel 80386, for MS Windows
user@machine$
```

There is a folder named Samples on the Desktop in the attached VM. We will be using the samples present in that folder for our analysis. The above terminal shows the file command being run on the 'wannacry' sample. The output shows a PE32 executable file with a Graphical User Interface, which was compiled for a system that runs Microsoft Windows with an Intel 80386-based processor. The Intel 80386 processor was one of the first 32-bit processors ever, and the instruction set designed for the 80386 is still used for 32-bit Intel processors, which is why you see "x86" processors and code. This means that the "80386" in the output above tells us that this application was designed for 32-bit Intel processors.

Examining Strings

Another really important command that provides us with useful information about a file is the strings command. This command lists down the strings present in a file. To understand what the string command does, we can read its man page or use the --help option:

```
man strings or strings --help
```

We will find that it is also a simple command to use. We can use the following command to find the strings in a file:

```
strings <filename>
```

Looking at strings in a file can often give clues related to the behavior of malware. For example, if we see URLDownloadToFile in the output of the strings command, we will know that this malware is doing something with the URLDownloadToFile Windows API. Most likely, it is downloading a file from the internet and saving it on the disk. Similarly, strings might also provide contextual information that helps us later during malware analysis.

Remnux

```
user@machine$ strings wannacry
!This program cannot be run in DOS mode.
Rich
.text
`.rdata
@.data
```



```
.rsrc
49t$
TVWj
PVVh
tE9u
.
.
.
.
  inflate 1.1.3 Copyright 1995-1998 Mark Adler
n;^
Qkkbal
i]Wb
9a&g
MGiI
wn>Jj
#.zf
+o*7
- unzip 0.15 Copyright 1998 Gilles Vollant
CloseHandle
GetExitCodeProcess
TerminateProcess
WaitForSingleObject
CreateProcessA
GlobalFree
GetProcAddress
LoadLibraryA
GlobalAlloc
SetCurrentDirectoryA
GetCurrentDirectoryA
GetComputerNameW
SetFileTime
SetFilePointer
MultiByteToWideChar
GetFileAttributesW
GetFileSizeEx
.
.
.
.
user@machine$
```

Here we can see the strings command being run against the 'wannacry' sample. We will see that the output starts with the DOS Stub, which is the text that says !This program cannot be run in DOS mode. Some values don't make much sense and look like garbage, but you will also see useful output. For example, we can see above that some strings look like Windows APIs. For example, CloseHandle, GetExitCodeProcess, TerminateProcess, and so on. Similarly, we can see text that says inflate 1.1.3 Copyright 1995-1998 Mark Adler. A quick search shows that it is a part of the zlib data compression library, this tells us that the sample might be using this library.

Tip: Sometimes, the output of the strings command is too big to be shown on the terminal completely. We can redirect it, write it to a file, and read it using vim or any other tool. The below terminal shows the output being redirected to a file named str:

Remnux

```
user@machine$ strings wannacry>str
user@machine$
```

Alternatively, you can use the more or less command to parse the output in a more visible manner:

Remnux

```
user@machine$ strings wannacry |more
!This program cannot be run in DOS mode.
Rich
.text
`.rdata
@.data
.rsrc
49t$
TVWj
PVVh
tE9u
PVVW
SVWjcf
X_^[
X_^[
^t19
QPPh
```

```
tXVP
X_^]
^t)9
X_^[]
WWWWWPj
SjJ3
X[_^
Yu#j
uSh8
Yu8S
SSh
hn!@
SVWj@
--more--
```

We can use the space key to scroll down the list of strings here. If you are interested, this room contains more information about strings.

Calculating Hashes

File Hashing provides us with a fixed-size unique number that identifies a file. A File Hash can therefore be considered a unique identifier for a file, similar to Social Security Numbers or National Identification Numbers used for the citizens of a country. Hashing is an important concept in malware analysis. It can be used as an identifier for specific malware. As we will see later in this task, this identifier can then be shared with other analysts or searched online for information sharing purposes. Please note that a single bit of difference in two files will result in different hashes, so changing the hash of a file is as simple as changing one bit in it.

Commonly, md5sum, sha1sum and sha256sum hashes are used for file hashing. We can calculate file hashes by using a simple command in Linux, as shown below for the md5sum hash:

```
md5sum <filename>
Remnux

user@machine$ md5sum wannacry
84c82835a5d21bbcf75a61706d8ab549  wannacry
user@machine$
```

Above, we can see the md5sum hash is calculated for the file named 'wannacry.'

Similarly, sha1sum and sha256sum commands can be used for calculating sha1sum or sha256sum of a file (Hashes are often referred to without the `sum` at the end, e.g. md5 instead of md5sum and so on.)

If you are interested in learning more about hashes, you can check out this room.
AV scans and VirusTotal

Scanning a file using AVs or searching for a hash on VirusTotal can also provide useful information about the classification of malware performed by security researchers. However, when using an online scanner, it is recommended to search for the malware's hash instead of uploading online to avoid leaking sensitive information online. Only upload a sample if you are sure of what you are doing.

Let's see what it says about the sample we calculated the hash for above. We can search for the md5sum we calculated for the wannacry sample on the VirusTotal homepage:

VirusTotal has a mix of handy features. It provides scan results from 60+ AV vendors and each AV vendor's classification to the sample.

The details tab lists the history of the sample, the first submission, the last submission, and the metadata of the sample.

Sometimes it also provides information about the behavior of a sample and its relations as seen in different environments online.

We can also find comments about the sample by the community on VirusTotal, which can sometimes provide additional context about the sample.

Perhaps it is very clear from the above screenshots that we are looking at a sample of wannacry ransomware.

Answer the questions below

In the attached VM, there is a sample named 'redline' in the Desktop/Samples directory. What is the md5sum of this sample?

```

ubuntu@ip-10-10-241-26:~/Desktop$ ls
Samples
ubuntu@ip-10-10-241-26:~/Desktop$ cd Samples/
ubuntu@ip-10-10-241-26:~/Desktop/Samples$ ls
redline  str  wannacry  zmsuz3pinwl
ubuntu@ip-10-10-241-26:~/Desktop/Samples$ md5sum redline
ca2dc5a3f94c4f19334cc8b68f256259  redline
ubuntu@ip-10-10-241-26:~/Desktop/Samples$ stat redline

```

ca2dc5a3f94c4f19334cc8b68f256259

What is the creation time of this sample?

I tried by using the command stat to find the birth time but it is showing null , so i searched the file hash in virustotal

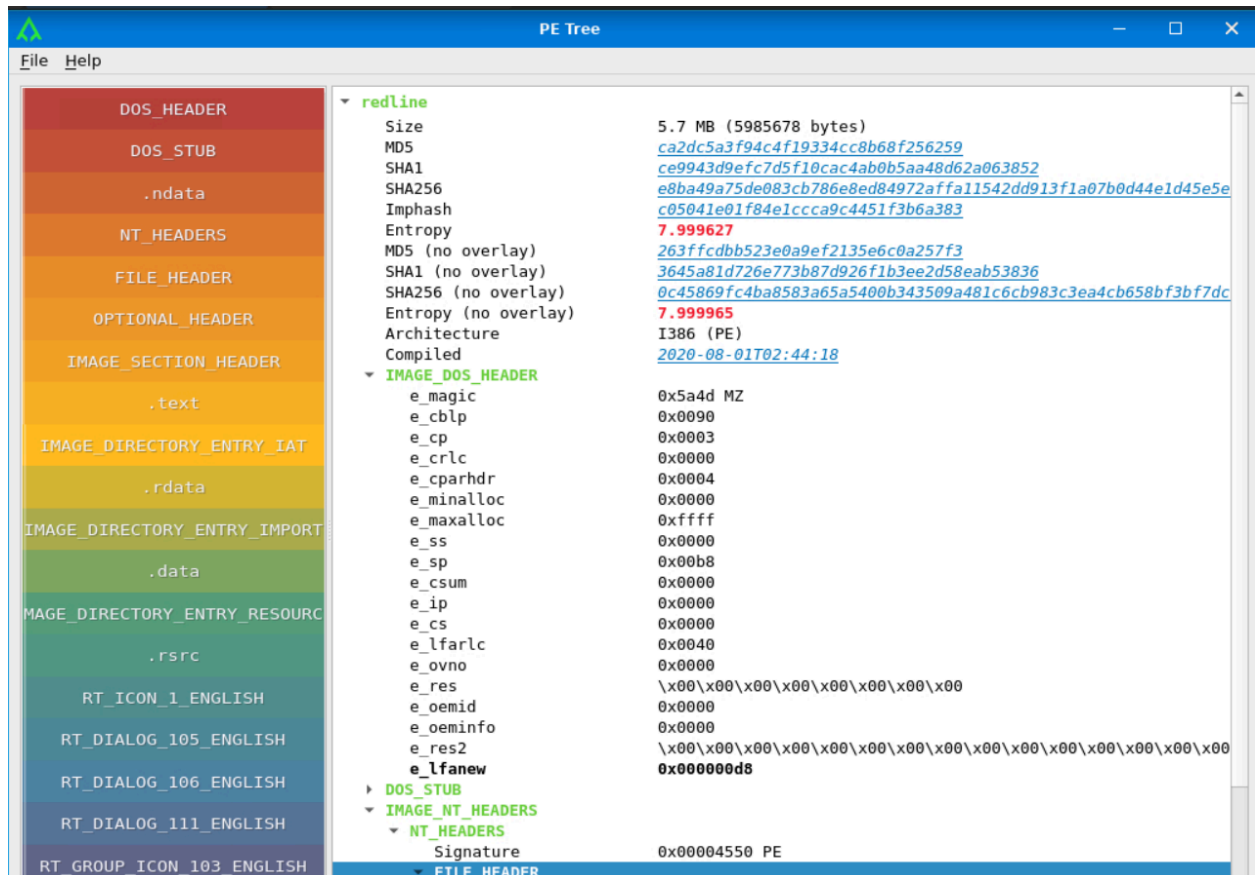
Click on the details tab scroll till history

History ⓘ	
Creation Time	2020-08-01 02:44:18 UTC
First Submission	2022-03-05 15:45:53 UTC
Last Submission	2025-10-18 01:13:58 UTC
Last Analysis	2025-10-19 02:38:38 UTC

2020-08-01 02:44:18 UTC

Task 5 The PE file Header

PE Header



The PE File Header contains the metadata about a Portable Executable file. This data can help us find a lot of helpful information to help us in our analysis. We will go into detail about the PE header and the information it contains in the upcoming Malware Analysis module for the advanced path. However, some of the vital information found in the PE header is explained below:

Imports/Exports

A PE file seldom contains all the code that it needs to run on a system on its own. Most of the time, it re-uses code provided by the Operating System. This is done to use less space and leverage the framework the Operating System has laid to perform tasks instead of re-inventing the wheel. Imports are such functions that the PE file imports from outside to perform different tasks.

For example, if a developer wants to Query a Windows Registry value, they will import the [RegQueryValue](#) function provided by Microsoft instead of writing the code themselves. It is understood that this function will be present on any Windows machine on which the developer's code is going to run, so it does not need to be included in the PE file itself. Similarly, any PE file export functions are exposed to other binaries that can use that function instead of

implementing it themselves. Exports are generally associated with Dynamically-Linked libraries (DLL files), and it is not typical for a non-DLL PE file to have a lot of exports.

Since most PE files use the Windows API to perform the bulk of their jobs, a PE file's imports provide us with crucial information on what that PE file will do. It becomes evident that a PE file that is importing the InternetOpen function will communicate with the internet, a URLDownloadToFile function shows that a PE file will download something from the internet, and so on. Names of Windows APIs are generally intuitive and self-explanatory. However, we can always consult [Microsoft Documentation](#) to verify the purpose of a particular Windows function.

Sections:

Another useful piece of information available in the PE file header is the information about sections in the PE file. A PE file is divided into different sections which have different purposes. Although the sections in a PE file depend on the compiler or packer used to compile or pack the binary, the following are the most commonly seen sections in a PE file.

- **.text:** This Section generally contains the CPU instructions executed when the PE file is run. This section is marked as executable.
- **.data:** This Section contains the global variables and other global data used by the PE file.
- **.rsrc:** This Section contains resources that are used by the PE file, for example, images, icons, etc.

Analyzing PE header using pecheck utility

We can use the pecheck utility present in the Remnux VM attached with the room to check the PE header.

```
Remnux
user@machine$ pecheck wannacry
PE check for 'wannacry':
Entropy: 7.995471 (Min=0.0, Max=8.0)
MD5      hash: 84c82835a5d21bbc75a61706d8ab549
SHA-1    hash: 5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
SHA-256  hash:
ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
SHA-512  hash:
90723a50c20ba3643d625595fd6be8dcf88d70ff7f4b4719a88f055d5b3149a4231018ea30d
375171507a147e59f73478c0c27948590794554d031e7d54b7244
.text entropy: 6.404235 (Min=0.0, Max=8.0)
.rdata entropy: 6.663571 (Min=0.0, Max=8.0)
```

```

.data entropy: 4.455750 (Min=0.0, Max=8.0)
.rsrc entropy: 7.999868 (Min=0.0, Max=8.0)
Dump Info:
-----DOS_HEADER-----

[ IMAGE_DOS_HEADER ]
0x0      0x0    e_magic:                0x5A4D
0x2      0x2    e_cblp:                0x90
0x4      0x4    e_cp:                  0x3
.
.
.
.
.
[ IMAGE_IMPORT_DESCRIPTOR ]
0xD5D0    0x0    OriginalFirstThunk:    0xD60C
0xD5D0    0x0    Characteristics:      0xD60C
0xD5D4    0x4    TimeDateStamp:        0x0          [Thu Jan 1
00:00:00 1970 UTC]
0xD5D8    0x8    ForwarderChain:      0x0
0xD5DC    0xC    Name:                0xDC84
0xD5E0    0x10   FirstThunk:          0x8000

ADVAPI32.dll.CreateServiceA Hint[100]
ADVAPI32.dll.OpenServiceA Hint[431]
ADVAPI32.dll.StartServiceA Hint[585]
ADVAPI32.dll.CloseServiceHandle Hint[62]
ADVAPI32.dll.CryptReleaseContext Hint[160]
ADVAPI32.dll.RegCreateKeyW Hint[467]
ADVAPI32.dll.RegSetValueExA Hint[516]
ADVAPI32.dll.RegQueryValueExA Hint[503]
ADVAPI32.dll.RegCloseKey Hint[459]
ADVAPI32.dll.OpenSCManagerA Hint[429]

```

Here we can see information pecheck has extracted from the PE header of the wannacry sample. We see that the sample has 4 sections, .text, .rdata, .data and .rsrc and their respective entropy. Similarly, it has also shown us the different hashes of the sample. Pecheck also shows us the functions that a PE file imports. In the above terminal window, we can see the IMAGE_IMPORT_DESCRIPTOR, which shows the functions it imports from the ADVAPI32.dll Linked library. We will see similar descriptors for all the other linked libraries whose functions are imported by the sample.

We can see that pecheck shows us a lot more information than what we discussed in this task; however, discussing all that information is out of the scope of this room. We will dive into further details in the upcoming malware analysis module. We will take what we are looking for from the information we see, namely, the section information and the imports of our samples.

Answer the questions below

In the attached VM, there is a sample named 'redline' in the directory Desktop/Samples.

What is the entropy of the .text section of this sample?

Execute pecheck redline

```
PE check for 'redline':
Entropy: 7.999627 (Min=0.0, Max=8.0)
MD5      hash: ca2dc5a3f94c4f19334cc8b68f256259
SHA-1    hash: ce9943d9efc7d5f10cac4ab0b5aa48d62a063852
SHA-256  hash: e8ba49a75de083cb786e8ed84972affa11542dd913f1a07b0d44e1d45e5e22e9
SHA-512  hash: 8c774f64631342c2465d166cd4c374356c40c1cf6bae13b2e0b003ce6c85e397da
799f111cbbbed638d548029c555f31156c2633d531fa1b20160d7904fa17d75
.text entropy: 6.453919 (Min=0.0, Max=8.0)
.rdata entropy: 5.136718 (Min=0.0, Max=8.0)
.data entropy: 4.096809 (Min=0.0, Max=8.0)
.ndata entropy: 0.000000 (Min=0.0, Max=8.0)
.rsrc entropy: 4.209687 (Min=0.0, Max=8.0)
```

6.453919

The sample named 'redline' has five sections. .text, .rdata, .data and .rsrc are four of them. What is the name of the fifth section?

.ndata from the above screenshot

From which dll file does the sample named 'redline' import the RegOpenKeyExW function?

```
ADVAPI32.dll.RegCreateKeyExW Hint[466]
ADVAPI32.dll.RegEnumKeyW Hint[480]
ADVAPI32.dll.RegQueryValueExW Hint[504]
ADVAPI32.dll.RegSetValueExW Hint[517]
ADVAPI32.dll.RegCloseKey Hint[459]
ADVAPI32.dll.RegDeleteValueW Hint[473]
ADVAPI32.dll.RegDeleteKeyW Hint[471]
ADVAPI32.dll.AdjustTokenPrivileges Hint[28]
ADVAPI32.dll.LookupPrivilegeValueW Hint[336]
ADVAPI32.dll.OpenProcessToken Hint[428]
ADVAPI32.dll.SetFileSecurityW Hint[559]
ADVAPI32.dll.RegOpenKeyExW Hint[493]
ADVAPI32.dll.RegEnumValueW Hint[482]
```

Scroll till the dlls visible

ADVAPI32.dll

Task 6 Basic Dynamic Analysis

While basic static analysis provides us with useful information about a sample, most times, we need to perform additional analysis to move further in our analysis procedure. One quick and dirty way to find more clues about a malware's behavior is by performing basic dynamic analysis. Many of the properties of a malware sample can be hidden when it's not running. However, when we perform dynamic analysis, we can lay these properties bare and learn more about the behavior of a malware sample.

Caution!

Dynamic analysis requires running live malware samples that can be destructive. It is highly recommended that you perform malware analysis in an isolated Virtual Machine. You can create a clean snapshot of your Virtual Machine before performing any malware analysis and revert it to start from a clean state again after every analysis. Don't perform malware analysis on a live machine not purpose-built for malware analysis.

Introduction to Sandboxes



Sandbox is a term borrowed from the military. A sandbox is a box of sand, as the name suggests, modeling the terrain where an operation has to take place, in which a military team dry runs their scenarios to identify possible outcomes. In malware analysis, a sandbox is an isolated environment mimicking the actual target environment of a malware, where an analyst runs a sample to learn more about it. Malware analysis sandboxes heavily rely on Virtual Machines, their ability to take snapshots and revert to a clean state when required.

Construction of a sandbox

For malware analysis using sandboxes, the following considerations make the malware analysis effective:

- Virtual Machine mimicking the actual target environment of the malware sample
- Ability to take snapshots and revert to clean state
- OS monitoring software, for example, Procmon, ProcExplorer or Regshot, etc.
- Network monitoring software, for example, Wireshark, tcpdump, etc.
- Control over the network through a dummy DNS server and webserver.
- A mechanism to move analysis logs and malware samples in and out of the Virtual Machine without compromising the host (Be careful with this one. If you have a shared directory with your malware analysis VM that remains accessible when running malware, you might risk malware affecting all files in your shared directory)

Open Source Sandboxes

Though it is good to understand what a good sandbox is made of, building a sandbox from scratch is not always necessary. One can always set up Open Source Sandboxes. These sandboxes provide the framework for performing basic dynamic analysis and are also customizable to a significant extent to help those with a more adventurous mindset.

Cuckoo's Sandbox

[Cuckoo's sandbox](#) is the most widely known sandbox in the malware analysis community. It was developed as part of a Google Summer of Code project in 2010. It is an open-source project that you will often see deployed in SOC environments and with enthusiasts' home labs. Advantages of Cuckoo's sandbox include huge community support, easy-to-understand documentation, and lots of customizations. You can deploy it on your network and let the community signatures guide you into identifying which files are malicious and which are benign because of the vast corpus of community signatures that come with it.

Cuckoo's sandbox has been archived, and an update is pending. It also doesn't support Python 3, making it obsolete right now. However, all is not lost because we have alternatives.

CAPE Sandbox

[CAPE Sandbox](#) is a little more advanced version of Cuckoo's sandbox. It supports debugging and memory dumping to support the unpacking of packed malware (We will learn more about packing and unpacking in the advanced malware analysis module). Though beginners can use this sandbox, advanced knowledge is required for making full use of it. A community version of this sandbox is available online, which can be used to test run it before installing. CAPE Sandbox is so far actively developed and supports Python 3.

Online Sandboxes:


Setting up and maintaining a sandbox can be a time-consuming task. Keeping that in view, online sandboxes can be of great help. Some of the most commonly used online sandboxes are as follows:

- [Online Cuckoo Sandbox](#)
- [Any.run](#)
- [Intezer](#)
- [Hybrid Analysis](#)

Though online sandboxes provide a useful utility, it is best not to submit a sample online unless you are sure of what you are doing. A better approach is to search for the sample's hash on the service you are using to see if someone has already submitted it. Let's look at Hybrid Analysis to see what interesting analysis it provides for our sample.

Analyzing samples using Hybrid Analysis

On its homepage, we are greeted with the following screen:



File/URL File Collection Report Search YARA Search String Search

Search through 15M+ Indicators of Compromise (IOCs).

Search

IP, Domain, Hash...

Search

or

Advanced Search

As we mentioned, we will not be submitting a sample. Instead, we will search for the hash of our sample. Therefore, we will search for the md5sum of the wannacry sample from the attached VM. We will see that it is already submitted multiple times, and we can choose from the submitted results.

Search results for <i>84c82835a5d21bbcf75a61706d8ab549</i>						<div>Multi-ProcessExtracted FilesSample not sharedNetwork TrafficTOR analysisDecrypted SSL traffic</div>	
Timestamp	Input	Threat level	Analysis Summary	Countries	Environment		
March 2nd 2022 05:13:13 (UTC)	.EXE PE32 executable (GUI) Intel 80386, for MS Windows ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa	malicious	AV Detection: 95% Trojan.Generic #tag #wannacry #Worm #ransomware #wannacryptOr #wcry #gozi #isfb #papras #ursnif	-	quicksan		
June 28th 2021 15:07:15 (UTC)	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.e PE32 executable (GUI) Intel 80386, for MS Windows ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa	malicious	Threat Score: 100/100 AV Detection: 95% Trojan.Generic Matched 79 Indicators #tag #wannacry #Worm #ransomware #wannacryptOr #wcry #gozi #isfb #papras #ursnif Show Similar Samples		Windows 7 32 bit		
May 15th 2021 00:17:17 (UTC)	FreePass.exe PE32 executable (GUI) Intel 80386, for MS Windows ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa	malicious	Threat Score: 100/100 AV Detection: 95% Trojan.Generic Matched 79 Indicators #tag #wannacry #Worm #ransomware #wannacryptOr #wcry #gozi #isfb #papras #ursnif Show Similar Samples		Windows 7 64 bit		

Let's open the one submitted on Windows 7 64 bit from among these.

owo_im_not_ransomware_xd.exe

This report is generated from a file or URL submitted to this webservice on February 24th 2020 16:03:53 (UTC)
Guest System: Windows 7 64 bit, Professional, 6.1 (build 7601), Service Pack 1
Report generated by Falcon Sandbox v8.30 © Hybrid Analysis

OverviewSample unavailableDownloadsExternal ReportsRe-analyzeHash Seen BeforeShow Similar SamplesRequest Report Deletion

malicious

Threat Score: 100/100
AV Detection: 95%
Labeled as: Trojan.Generic

#tag#wannacry#worm#ransomware#wanacryptor#wcry

LinkTwitterE-Mail

Incident Response

Related Sandbox Artifacts

Indicators

File Details

Screenshots (7)

Hybrid Analysis (34)

Network Analysis

Extracted Strings

Extracted Files (2000)

Notifications

Community (50)

Back to top

Incident Response

Risk Assessment

Remote Access

Reads terminal service related keys (often RDP related)

Ransomware

Deletes volume snapshots (often used by ransomware)
Detected indicator that file is ransomware

Spyware

Contains ability to open the clipboard
Deletes volume snapshots (often used by ransomware)

Persistence

Grants permissions using iccls (DACL modification)
Spawns a lot of processes
Tries to suppress failures during boot (often used to hide system changes)
Writes data to a remote process

Fingerprint


Queries kernel debugger information
Queries process information

We will see the above interface when we click on the sample. We can see a navigation pane on the right that highlights different parts of the report. We can also see that the verdict is malicious, with a threat score of 100/100 and AV detection of 95%. Below that, we see the overview of the sample's behavior. Below that, we will see the mapping to MITRE ATT&CK techniques. We will see the following mapping when we click :

MITRE ATT&CK™ Techniques Detection						
Execution						
ATT&CK ID	Name	Tactics	Description	Malicious Indicators	Suspicious Indicators	Informative Indicators
T1047	Windows Management Instrumentation	<ul style="list-style-type: none">Execution	Windows Management Instrumentation (WMI) is a Windows administration feature that provides a uniform environment for local and remote access to Windows system components. Learn more		<ul style="list-style-type: none">Reads system information using Windows Management Instrumentation Commandline (WMIC)	<ul style="list-style-type: none">Contains references to WMI/WMICExecutes WMI queries
Persistence						
ATT&CK ID	Name	Tactics	Description	Malicious Indicators	Suspicious Indicators	Informative Indicators
T1044	File System Permissions Weakness	<ul style="list-style-type: none">PersistencePrivilege Escalation	Processes may automatically execute specific binaries as part of their functionality or to perform other actions. Learn more	<ul style="list-style-type: none">Modifies the access control lists of files	<ul style="list-style-type: none">Grants permissions using iccls (DACL modification)	
T1179	Hooking	<ul style="list-style-type: none">Credential AccessPersistencePrivilege Escalation	Windows processes often leverage application programming interface (API) functions to perform tasks that		<ul style="list-style-type: none">Installs hooks/patches the running process	<ul style="list-style-type: none">Loads rich edit control libraries
				Download as CSV Close		

Below that, we will see some indicators and context information and some static analysis information for the sample. The dynamic analysis part comes below that:

Hybrid Analysis

 Tip: Click an analysed process below to view more details.

Analysed 34 processes in total (System Resource Monitor).

- owowim_not_ransomware_xd.exe (PID: 3792) 75/83
 - attrib.exe attrib +h . (PID: 3820) Hash Seen Before
 - icacls.exe icacls . /grant Everyone:F /T /C /Q (PID: 2036) Hash Seen Before
 - taskd.exe (PID: 2120) 58/68 Hash Seen Before
 - cmd.exe %WINDIR%\system32\cmd.exe /c 103811582560339.bat (PID: 3532) Hash Seen Before
 - cscript.exe //nologo m.vbs (PID: 3852) Hash Seen Before
 - attrib.exe attrib +h +s %SAMPLEDIR%\\$RECYCLE (PID: 3804) Hash Seen Before
 - taskd.exe (PID: 3396) 58/68 Hash Seen Before
 - taskd.exe (PID: 3524) 58/68 Hash Seen Before
 - taskd.exe (PID: 1552) 58/68
 - @WanaDecryptor.exe co (PID: 3968) 74/82 Hash Seen Before
 - taskhsvc.exe TaskData\Tor\taskhsvc.exe (PID: 3356) Hash Seen Before
 - cmd.exe /c start /b @WanaDecryptor.exe vs (PID: 3384) Hash Seen Before
 - @WanaDecryptor.exe vs (PID: 2576) 74/82 Hash Seen Before
 - cmd.exe /c vssadmin delete shadows /all /quiet & wmic shadowcopy delete & bcdedit /set (default) bootstatuspolicy ignoreallfailures & bcdedit /set (default) recoveryenabled no & wbadmin delete catalog -quiet (PID: 3868) Hash Seen Before
 - vssadmin.exe vssadmin delete shadows /all /quiet (PID: 3020) Hash Seen Before
 - WMIC.exe wmic shadowcopy delete (PID: 3588) Hash Seen Before
 - taskse.exe C:\@WanaDecryptor.exe (PID: 3136) 59/69 Hash Seen Before
 - @WanaDecryptor.exe (PID: 2296) 74/82 Hash Seen Before

This part provides us with a lot of information about the behavior of the sample when it was run in a sandbox. We can click each process to find more detail about it. In the above screenshot, of particular interest can be the executions of cmd.exe. We can see that the sample is running script files and deleting backups and volume shadow copies, something often done by ransomware operators to stop the victim from restoring their files from these sources.

Below this section, we will see network analysis of the sample:













Network Analysis

DNS Requests

No relevant DNS requests were made.

Contacted Hosts

Login to Download Contacted Hosts (CSV)

IP Address	Port/Protocol	Associated Process	Details
85.235.250.88 	443 TCP	taskhsvc.exe PID: 3356	 Denmark
194.109.206.212 	443 TCP	taskhsvc.exe PID: 3356	 Netherlands
188.40.128.246 	9001 TCP	taskhsvc.exe PID: 3356	 Germany
212.47.244.38 	443 TCP	taskhsvc.exe PID: 3356	 France
154.35.175.225 	443 TCP	taskhsvc.exe PID: 3356	 United States
212.47.233.86 	9001 TCP	taskhsvc.exe PID: 3356	 France

Extracted strings and extracted files are also available in the report. These can provide information about the batch scripts we saw in the processes above.

Extracted Strings

All Details:

[Download All Memory Strings \(57KiB\)](#)

[All \(3589\)](#) [106921521127277.bat \(2\)](#) [@WanaDecryptor.exe:214...](#) [@WanaDecryptor.exe:363...](#) [@WanaDecryptor.exe:419...](#) [WMIC.exe:1636 \(226\)](#) [WannaCry.EXE:1608 \(6\)](#)
[WannaCry.EXE.bin \(267\)](#) [attrib.exe \(1\)](#) [attrib.exe:2300 \(1\)](#) [attrib.exe:3240 \(2\)](#) [cached-microdesc-consens...](#) [cmd.exe \(4\)](#) [cscript.exe \(1\)](#) [icacds.exe:3956 \(1\)](#)
[libeay32.dll.143134089 \(2\)](#) [libevent-2-0-5.dll.2395423...](#) [libevent_core-2-0-5.dll.417...](#) [libgcc_s_sjlj-1.dll.413159514...](#) [network.pcap \(209\)](#) [reg.exe:908 \(2\)](#) [screen_0.png \(5\)](#)
[screen_3.png \(9\)](#) [screen_5.png \(74\)](#) [taskdel.exe:3392 \(17\)](#) [taskhsvc.exe:2060 \(2808\)](#) [taskse.exe \(1\)](#) [tor.exe.2844654368 \(3\)](#) [vssadmin.exe:3688 \(2\)](#)

echo off
echo offecho SET ow = WScript.CreateObject("WScript.Shell")> m.vbsecho SET om = ow.CreateShortcut("C:\@WanaDecryptor.exe.lnk")>> m.vbsecho om.TargetPath = "C:\@WanaDecryptor.exe">> m.vbsecho om.Save>> m.vbscscript.exe //nologo m.vbsdel m.vbsdel /a %0

Extracted Files

Displaying 22 extracted file(s). The remaining 1978 file(s) are available in the full version and XML/JSON reports.

And there are comments from the community at the very end. As we might have seen, we can find many pieces of the puzzle that a malware sample is, using the discussed techniques. However, in some cases, these techniques can prove insufficient to make a decision. Let's move to the next task to determine what scenarios can make it challenging to analyze malware.

Answer the questions below



Check the hash of the sample 'redline' on Hybrid analysis and check out the hybrid analysis report. In the process tree, which is the first process launched when the sample is launched?

Take the md5 hash of the file , search the hash in hybrid analysis , click on any report , scroll till the hybrid analysis section

Hybrid Analysis

 **Tip:** Click an analysed process below to view more details.

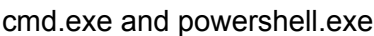
Analysed 43 processes in total (System Resource Monitor).

- redline.exe (PID: 2900)   Hash Seen Before
 - setup_installer.exe (PID: 3180)   Hash Seen Before
 - setup_installer.exe (PID: 2820)   Hash Seen Before
 - cmd.exe %WINDIR%\system32\cmd.exe /c powershell -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath "%TEMP%\ (PID: 1216)  Hash Seen Before
 - powershell.exe powershell -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath "%TEMP%\ (PID: 3172)  Hash Seen Before
 - cmd.exe %WINDIR%\system32\cmd.exe /c 62237fa25bdfc_Sat15d93b81243b.exe (PID: 3464)  Hash Seen Before
 - 62237fa25bdfc_Sat15d93b81243b.exe (PID: 2888)   Hash Seen Before
 - cmd.exe %WINDIR%\system32\cmd.exe /c powershell -inputformat none -outputformat none -NonInteractive -Command Set-MpPreference -DisableRealtimeMonitoring \$true -SubmitSamplesConsent NeverSend -MAPSReporting Disable (PID: 2188)  Hash Seen Before
 - powershell.exe powershell -inputformat none -outputformat none -NonInteractive -Command Set-MpPreference -DisableRealtimeMonitoring \$true -SubmitSamplesConsent NeverSend -MAPSReporting Disable (PID: 2404)  Hash Seen Before
 - cmd.exe %WINDIR%\system32\cmd.exe /c 62237fa38bcf4_Sat15f98352f48.exe (PID: 3076)  Hash Seen Before
 - 62237fa38bcf4_Sat15f98352f48.exe (PID: 2324)   Hash Seen Before
 - cmd.exe %WINDIR%\system32\cmd.exe /c 62237fa44c09c_Sat15f6e00f22a4.exe (PID: 3460)  Hash Seen Before
 - 62237fa44c09c_Sat15f6e00f22a4.exe (PID: 2244)   Hash Seen Before
 - 62237fa44c09c_Sat15f6e00f22a4.exe (PID: 3612)   Hash Seen Before

setup_installer.exe

In the process tree, there are two Windows utilities utilized by the malware to perform its activities. What are the names of the two utilities? (Format: utility1.exe and utility2.exe)

 Tip: Click an analysed process below to view more details.



While the security researchers are devising techniques and tools to analyze malware, the malware authors are working on rendering these tools and techniques ineffective. We found a great deal of information in the previous tasks about the malware we analyzed. However, there are ways malware authors can make our life difficult. Below are some of the techniques used by malware authors to do the same.

Malware authors often use packing and obfuscation to make an analyst's life difficult. A packer obfuscates, compresses, or encrypts the contents of malware. These techniques make it difficult to analyze malware statically. Specifically, a packed malware will not show important information when running a string search against it. For example, let's run a string search against the file named `zmsuz3pinwl` in the Samples folder in the attached VM.

```
Remnux
      user@machine$ strings zmsuz3pinwl
!This program cannot be run in DOS mode.
RichH
.rsrc
.data
.adata
dApB
Qtq5
wn;3b:TC,n
*tVlr
D6j[
^sZ"4V
```

```
JIoL
j~AI
tYFu
7^V1
vYB09
"PeHy
M4AF#
3134
%}W\+
3A;a5
dLq<
```

We will notice that this sample contains mainly garbage strings that don't provide much value to us. Let's run pecheck on the sample to see what else we get.

```
Remnux
      user@machine$ pecheck zmsuz3pinwl
PE check for 'zmsuz3pinwl':
Entropy: 7.978052 (Min=0.0, Max=8.0)
MD5      hash: 1ebb1e268a462d56a389e8e1d06b4945
SHA-1    hash: 1ecc0b9f380896373e81ed166c34a89bded873b5
SHA-256  hash:
98c6cf0b129438ec62a628e8431e790b114ba0d82b76e625885ceedef286d6f5
SHA-512  hash:
6921532b4b5ed9514660eb408dfa5d28998f52aa206013546f9eb66e26861565f852ec7f04c
85ae9be89e7721c4f1a5c31d2fae49b0e7fd20451191146614a
  entropy: 7.999788 (Min=0.0, Max=8.0)
  entropy: 7.961048 (Min=0.0, Max=8.0)
  entropy: 7.554513 (Min=0.0, Max=8.0)
.rsrc entropy: 6.938747 (Min=0.0, Max=8.0)
  entropy: 0.000000 (Min=0.0, Max=8.0)
.data entropy: 7.866646 (Min=0.0, Max=8.0)
.adata entropy: 0.000000 (Min=0.0, Max=8.0)
Dump Info:
-----Parsing Warnings-----

Suspicious flags set for section 0. Both IMAGE_SCN_MEM_WRITE and
IMAGE_SCN_MEM_EXECUTE are set. This might indicate a packed executable.

Suspicious flags set for section 1. Both IMAGE_SCN_MEM_WRITE and
IMAGE_SCN_MEM_EXECUTE are set. This might indicate a packed executable.
```

Suspicious flags set for section 2. Both IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE are set. This might indicate a **packed** executable.

Suspicious flags set for section 3. Both IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE are set. This might indicate a **packed** executable.

Suspicious flags set for section 4. Both IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE are set. This might indicate a **packed** executable.

Suspicious flags set for section 5. Both IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE are set. This might indicate a **packed** executable.

Suspicious flags set for section 6. Both IMAGE_SCN_MEM_WRITE and IMAGE_SCN_MEM_EXECUTE are set. This might indicate a **packed** executable.

Imported symbols contain entries typical of **packed** executables.

As suspected, we see that the executable has characteristics typical of a packed executable, as per pecheck. We will notice that there is no .text section in the sample, and other sections have execute permissions, which shows that these sections contain executable instructions or will be populated with executable instructions during execution. We will also see that this sample does not have many imports that might show us its functionality, as we saw with the previous sample.

For analysis of packed executables, the first step is generally to unpack the sample. This is an advanced topic that will be covered in the upcoming rooms.

Sandbox evasion:

As we have seen previously, we can always run a sample in a sandbox to analyze it. In many cases, that might help us analyze samples that evade our basic static analysis techniques. However, malware authors have some tricks up their sleeves that hamper that effort. Some of these techniques are as follows:

- **Long sleep calls:** Malware authors know that sandboxes run for a limited time. Therefore, they program the malware not to perform any activity for a long time after execution. This is often accomplished through long sleep calls. The purpose of this technique is to time out the sandbox.
- **User activity detection:** Some malware samples will wait for user activity before performing malicious activity. The premise of this technique is that there will be no user in a sandbox. Therefore there will be no mouse movement or typing on the keyboard.

Advanced malware also detects patterns in mouse movements that are often used in automated sandboxes. This technique is designed to bypass automated sandbox detection.

- **Footprinting user activity:** Some malware checks for user files or activity, like if there are any files in the MS Office history or internet browsing history. If no or little activity is found, the malware will consider the machine as a sandbox and quit.
- **Detecting VMs:** Sandboxes run on virtual machines. Virtual machines leave artifacts that can be identified by malware. For example, some drivers installed in VMs being run on VMWare or Virtualbox give away the fact that the machine is a VM. Malware authors often associate VMs with sandboxes and would terminate the malware if a VM is detected.

The above list is not exhaustive but gives us an idea of what to expect when analyzing malware. In a future module dedicated to malware analysis, we will discuss these techniques and ways to detect malware that employs them.

Answer the questions below

Which of the techniques discussed above is used to bypass static analysis?

packing

Which technique discussed above is used to time out a sandbox?

long sleep calls