Malware Analysis 101 - Basic Static Analysis

Aditya Anand Follow
Sep 18 · 10 min read

> *This article is a continuation of my previous write-up "Malware Analysis 101", do give it a read before going ahead with this one to have a better understanding of the things that I will be explaining here.*

Malware Analysis is broadly divided into two groups Static Analysis & Dynamic Analysis. We can describe static analysis to be all those examinations of the malware where we don't actually execute the malware but try to figure out what the malware is trying to do and the commands it is attempting to execute. Dynamic analysis, on the other hand, is all those examinations that you carry out when you actually execute the malware most preferably in a sandboxed environment and then try to figure out the functionality of the malware.

Even though we have two well-defined methodologies we still have a further subdivision of these groups. Those being the Basic & Advanced methodologies that we use while trying to figure out the real motive behind these malware. In this article, I am trying to explain the basic static analysis methodologies of malware analysis.

Let's begin!

I was actually planning to focus on both basic static and basic dynamic analysis in this article but to properly explain everything I am sticking to basic static analysis. Now before going in depth regarding each of the technique that is used in basic static analysis let's discuss what it actually means.

Static analysis consists of examining the executable file without viewing the actual instructions. It is used to confirm, at least get an idea whether the file being inspected is malicious or not. We do this by figuring out the functions and libraries that are being called by the executable. Even though it is not much effective, a basic static analysis does act as a stepping stone for the rest of your malware analysis and gives you the idea about things you should be looking into.

Before diving into the static analysis methodologies first let us explore more about malware so that we understand all the steps that we take while doing analysis on software we deem to be malicious.

Packed & Obfuscated Malware

Malware is generally of two types those which are obfuscated and those which are not. The ones which aren't obfuscated can be very well analyzed by static tools but nowadays malware is mostly packed & obfuscated.

Obfuscated programs are ones whose execution the malware author has attempted to hide. *Packed* programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed. To identify if malware is packed or not we can carry a static check on it with Strings and if we find extremely few numbers of strings then there is a near 100% chance that the code is malicious.

- Packed and obfuscated code will at least include the functions like LoadLibrary and GetProcAddress, which are used to load and gain access to additional functions and are a give away indicating that the program is a malware.
- Packing Files

i) When the packed program is run, a small wrapper program also runs to decompress the packed file and then run the unpacked file

ii) Analyzing the packed program we can only make clear sense of the wrapper program and it can be then dissected, check the figure below.

The file on the left is the original executable with all strings, imports and other information visible. On the right hand is a packed executable. All of the packed file's strings, imports and other information are compressed and are invisible to most static analysis tools.

Now let's try to understand the portable executable file and their format so that we will have idea what to look for.

Portable Executable File Format

Portable executable file format is used by Windows executables, object codes, and DLLs. The PE file format is a data structure that contains the information necessary for the Windows OS loader describing how to manage the wrapped executable code.

- Nearly every file with executable code loaded by Windows is in the PE file format except for few of the legacy file formats which occur on rare occasions in malware.
- PE files begin with a header that includes

i) Information about the code

ii) Type of application

iii) Required library functions

iv) Space requirements

- The information received from the PE header can be of great value for the malware analyst

The Portable Executable File Headers and Sections

PE file headers can provide considerably more information than just imports. The PE file format contains a header followed by a series of sections. A lot of information can be derived from the header as it contains metadata about the file itself. Following the header file, we also have access the actual sections of the file, each of which contains useful information.

The following are the most common and interesting sections in a PE file:

- PE Header Summary

The PE header contains useful information for the malware analyst, and we will continue to examine it in subsequent chapters. Few of the key information that can be obtained from a PE header.

We know that the malware needs to use linked libraries & functions to work properly, so let's discover that.

Linked Libraries

One of the most useful pieces of information that we can gather about an executable is the list of linked libraries it utilizes.

- Code libraries can be linked statically, at runtime or dynamically

Knowing how the library code is linked it can be critical to our understanding of the malware as the information we find in the PE header can be made sense of depending on how these libraries interact and link with each other.

Static Linking

- It is the least commonly used method of linking libraries (common for UNIX & Linux)

- When statically linked, all code is copied into the executable which in turn increases the size of the file itself.
- Though analyzing the code does become difficult as it becomes extremely hard to differentiate between the statically linked code & the executable’s own code.
- If static linked is used, then the PE header files don’t indicate that the file contains linked code.

Runtime Linking

- It is used mainly in malware programs, especially when it’s packed or obfuscated.
- Executables that use this technique connect to libraries only when needed.
- Windows function allow programs to call functions not listed in the program’s header file.
- The two most commonly used functions are *LoadLibrary* and *GetProcAddress*
- *LdrGetProcAddress* and *LdrLoadDll* are also used
- *LoadLibrary* and *GetProcAddress* allow a program to access any function in any library on the system.
- So whenever these functions are used we can’t tell exactly which functions are being linked to the suspect program.

Dynamic Linking

- This is the most common method used with malwares
- When libraries are dynamically linked, the host OS searches for the necessary libraries whenever the program is loaded
- When the program calls the linked library function, that function executes within the library.
- The PE file header stores the information about every library that will be loaded and every function that will be used by the program
- Identifying the libraries being used are extremely important as it allows us to guess what the program is trying to do.
- Common DLLs that are mostly required and can be used to make certain deductions.

Common DLLs

Functions

Functions that are being called by the portable executable gives us in depth understanding about its workings. There are three different types of linked libraries and in the same way we also have two major sub division in functions, i.e. import and export.

Import Functions

- *Imports* are functions used by one program to link to code libraries the contains the required functionality and are stored in different programs.

- PE file header also includes information about specific functions used by an executable.
- Name of these Windows functions can give you a proper idea of what the executable does.

Export Functions

- DLLs and EXEs export functions to interact with other programs and code.
- DLL implements one or more functions and exports them for use by an executable that can then import and use them.
- PE file contains information about which functions a file exports.
- DLLs are specifically implemented to provide functionality used by EXEs
- If you discover exports in an executable, they often will provide useful information
- In many cases, software authors name their exported functions in a way that provides useful information.
- Therefore, the names of exported functions are actually of limited use against sophisticated malware.
- If malware uses exports, it will often either omit names entirely or use unclear or misleading names.

Now that we have covered the basics let’s start with the techniques used to do an effective basic static analysis.

1. Antivirus Scanning

This is the first step that you can carry out to figure out whether the particular program that you doubt to be malware is actually malicious or not. Most of the time the software that you want to check for security reasons might have already been identified by the major antivirus companies and it will save a hell lot of time for you trying to figure that out by yourself.

- These are not perfect by any means, they carry out the scan using already known suspicious code (file signatures), as well as behavior & pattern-matching analysis (heuristics).
- Significant changes in code can be used to bypass file signature check. Hence new and unique malware can bypass such heuristics check.
- Different antivirus tools use different signatures to figure out the malware, so it is recommended to test the malware file against various antivirus (virustotal.com).

2. Hashing

Hashing is a common technique that is used to uniquely identify malware. The thing about hashing is that once we have the unique hash of the software we deem to be malicious we can then use it for several purposes

i) Use it as a label to identify it across the malware analysis community

ii) Share it with other analysts to help them identify the malware

iii) Search for that malware online and check if it has been already identified

3. Finding strings

We know what string is, the strings that we are talking about are a sequence of characters that are present in the program. So in this particular technique, we try to fish out the strings that are present in the program, like a message, something connecting to a URL, or copies of file that might be present at a specific location, etc.

- Searching through these strings could be a way to get hints about the functionality of a program.
- Strings (bit.ly/ic4plL) tool by Microsoft can be used to search an executable for strings, which are typically stored in either ASCII or Unicode format
- Strings searches for a three-letter or greater sequence of ASCII and Unicode characters, followed by the termination character

> ***Note:*** *Both ASCII & Unicode formats stores characters in sequences that end with a NULLL terminator to indicate string completion*

- Strings ignore context and formatting while it searches an executable for ASCII & Unicode strings so that it can analyze any file type and detect strings across the entire file
- Not all strings found by Strings are valid strings they can be string, memory address, CPU instructions or data used by the program, leaves to the user to filter them out.

4. Detecting packers with PEiD

Using PEiD we can detect the type of packer or compiler employed to build an application, which makes analyzing the packed file much easier e.g. Id proof, photo, payment.

- The support and development on PEiD have been stopped in 2011 yet it is one of the best tools and in a few cases, it can also identify the packer used to pack the file.

PEid Software Output

> ***Note :*** *PEiD has identified the file as being packed with UPX version 0.89.6 - 1.02 / 1.05 - 2.90*

- We now know the method used to pack this, so we can now unpack this mostly they are extremely complex but lucky for us UPX packed malware can be easily unpacked, just download it from (upx.sourceforge.net/)

```
$ upx -d PackedProgram.exe
```

- Many PEiD plugins run the malware executable without warning! Like other programs, especially those used for malware analysis, PEiD can be subject to vulnerabilities.

> ***Note :*** *PEiD version 0.92 contained a buffer overflow that allowed an attacker to execute arbitrary code. This would have allowed a clever malware writer to write a program to exploit the malware analyst's machine. So prefer using only the latest version of PEiD.*

Other PE File Tools

Many other tools are available for browsing a PE header. Two of the most useful tools are PEBrowse Professional and PE Explorer.

PEBrowse Professional is similar to PEview. It allows you to look at the bytes from each section and shows the parsed data. PEBrowse Professional does the better job of presenting information from the resource (.rsrc) section.

PE Explorer has a rich GUI that allows you to navigate through the various parts of the PE file. You can edit certain parts of the PE file, and its included resource editor is great for browsing and editing the file's resources. The tool's main drawback is that it is not free.

Edit: Most of the stuff mentioned in the article and the screenshots are taken from the book -Practical Malware Analysis. You should definitely consider buying the book, link here. I have summarised my notes here and a few points have been directly picked from the book as I considered them to very well explained and it didn't make sense to rephrase or edit them.

If you enjoyed it please do clap & let's collaborate. Get, Set, Hack!

Website : aditya12anand.com | Donate : paypal.me/aditya12anand

Telegram : https://t.me/aditya12anand

Twitter : twitter.com/aditya12anand

LinkedIn : linkedin.com/in/aditya12anand/

E-mail : aditya12anand@protonmail.com

. . .

Follow Infosec Write-ups for more such awesome write-ups.