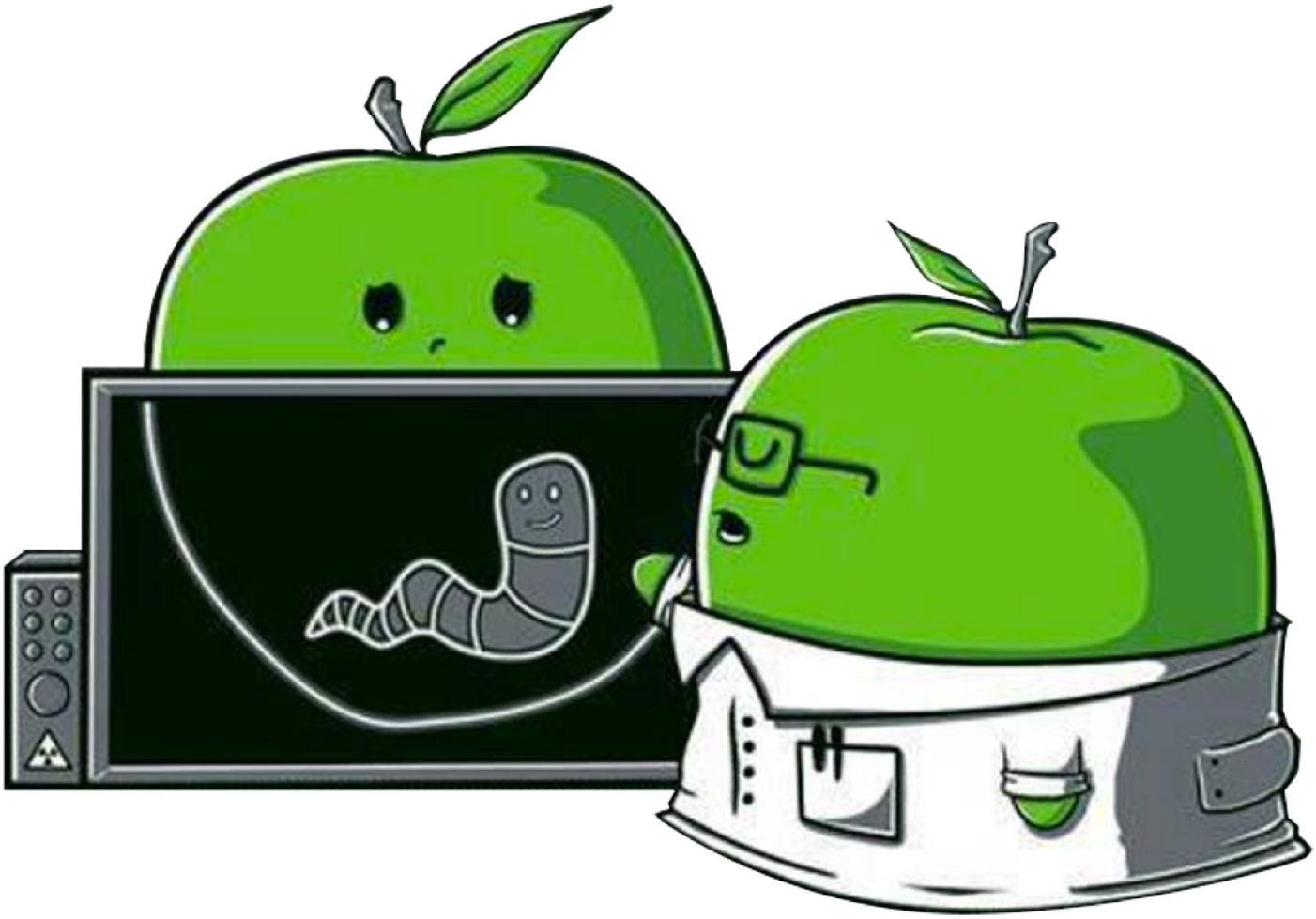


(The Art of Mac Malware) Volume 1: Analysis

Chapter 0x4: Static Analysis (Intro)

📝 Note:

This book is a work in progress.

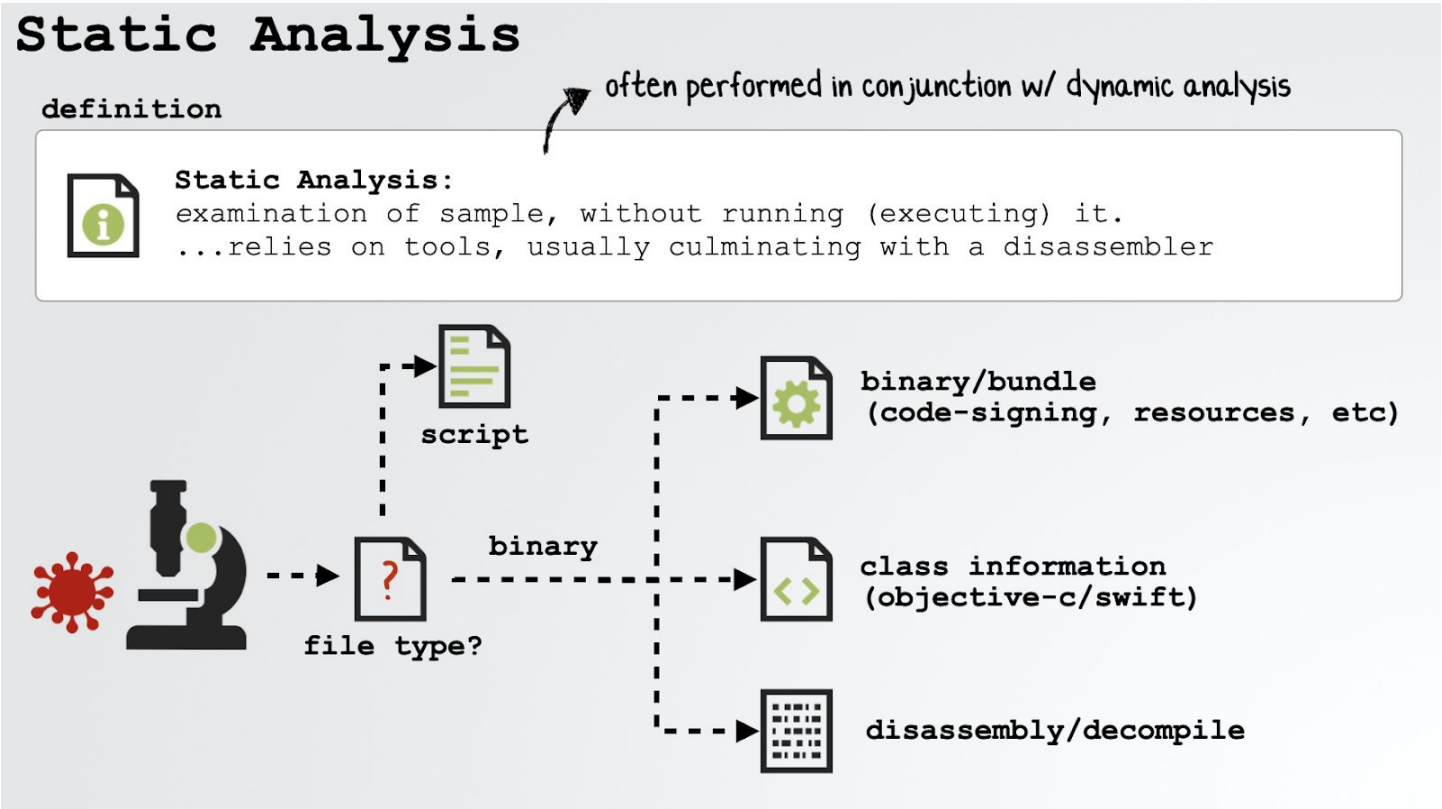
You are encouraged to directly comment on these pages ...suggesting edits, corrections, and/or additional content!

To comment, simply highlight any content, then click the icon which appears (to the right on the document's border).

Statically analyzing a (suspected) malware specimen involves examining the specimen without actually running or executing it. Such analysis relies on various static analysis tools and usually culminates with a disassembler or decompiler.

In this chapter, we'll comprehensively cover methods of static analysis, starting with the basics, such as file type identification and extraction from an installation medium.

Once a sample has been extracted (e.g. from a disk image or package), it's often in one of two forms: a script or a (Mach-O) binary.



the flow of static analysis

Because of their "plaintext readability," scripts are rather easy to manually analyze ...though we'll still cover various analysis techniques and apply them to real-world macOS malware samples.

On the other hand, the binary format of Mach-O executables presents some unique challenges and requires specific analysis tools. As such, a significant portion of this book is dedicated to both the internals of this file format and corresponding static analysis tools.

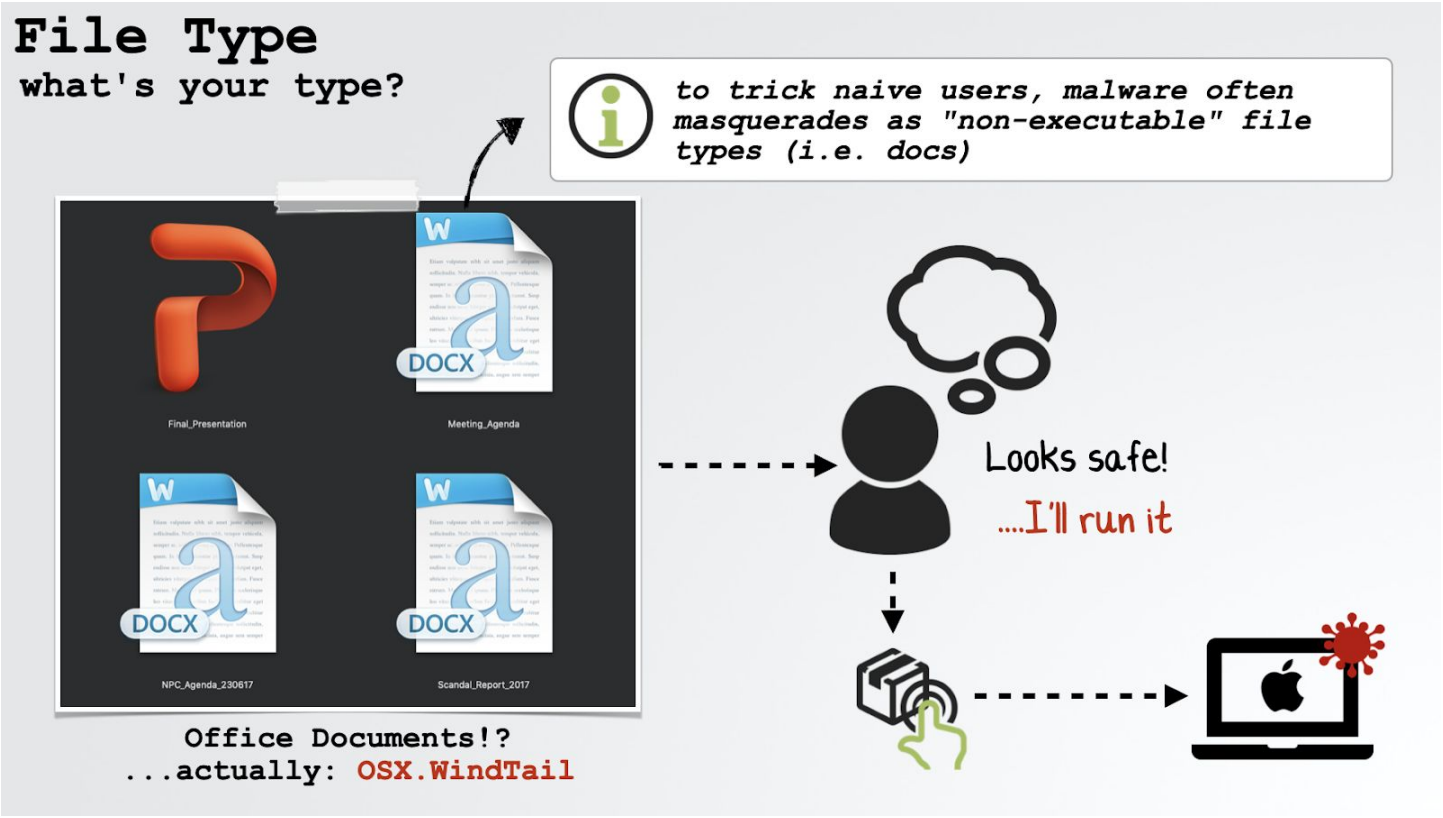
> 📝 Note:
>
> Is it safe to statically analyze malware on your computer (i.e. not in a virtual machine)?
>
> Generally yes, as by definition static analysis, is well, static ...meaning the malicious code is never run.
>
> That having been said, it is still considered best practice to always analyze malware in a compartmented virtual machine! Better safe than sorry, ya?
>
> For a detailed overview of setting up such a VM, see:
>
> "How to Reverse Malware on macOS Without Getting Infected" [1]

File Type Identification

As noted, most (static) analysis tools are file-type specific. Thus, the first step in analyzing (what may be) a malicious file is identifying its file type.

Often, malware authors will attempt to mask the true file type of their creation in order to trick or coerce the user into running it. As such, it goes without saying that looks can be deceiving and a file's type should not be identified solely by its appearance (icon) or what appears to be its file extension.

For example, OSX.WindTail [2] is specifically designed to masquerade as benign Microsoft Office documents:



malware masquerading as Office documents (`OSX.WindTail`)

In reality, the file(s) are malicious applications that, when executed, will persistently infect the system.

At the other end of the spectrum, malicious files may also have no icon, nor file extension. Moreover a cursory triage of the contents of such files may provide no clues about the file's actual type.

For example, here we have a (suspected) malicious file simply named `VtZkT` [3] ...of some unknown binary format:

```
$ hexdump -C VtZkT
00000000  03 f3 0d 0a 97 93 55 5b  63 00 00 00 00 00 00 00  |......U[c.......|
00000010  00 03 00 00 00 40 00 00  00 73 36 00 00 00 64 00  |.....@...s6...d.|
00000020  00 64 01 00 6c 00 00 5a  00 00 64 00 00 64 01 00  |.d..l..Z..d..d..|
00000030  6c 01 00 5a 01 00 65 00  00 6a 02 00 65 01 00 6a  |l..Z..e..j..e..j|
00000040  03 00 64 02 00 83 01 00  83 01 00 64 01 00 04 55  |..d........d...U|
00000050  64 01 00 53 28 03 00 00  00 69 ff ff ff ff 4e 73  |d..S(....i....Ns|
00000060  d8 08 00 00 65 4a 79 64  56 2b 6c 54 49 6a 6b 55  |....eJydV+lTIjkU|
00000070  2f 38 35 66 51 56 47 31  53 33 71 4c 61 52 78 6e  |/85fQVG1S3qLaRxn|
```

```
00000080  6e 42 6d 6e 4e 6c 73 4f  6c 2b 41 67 49 71 43 67  |nBmnNlsOl+AgIqCg|
00000090  4c 45 76 31 45 53 54 59  46 2b 6c 75 44 69 33 2f  |LEv1ESTYF+luDi3/|
000000a0  39 33 31 4a 4f 6b 32 72  75 47 50 74 46 7a 70 35  |931JOk2ruGPtFzp5|
```

Since static analysis tools are largely file type specific, identifying this file's type is imperative in order to continue static analysis. So, how do we effectively identify a file's format? Via macOS's built-in `file` command. As noted in its man page [4], this command has one job: to "*determine [a] file's type*":

```
$ man file

FILE(1)                   BSD General Commands Manual                  FILE(1)

NAME
     file -- determine file type
```

For example, running the `file` command on the unknown `VtZkT` file, reveals the file is byte-compiled Python code:

```
$ file VtZkT

VtZkT: python 2.7 byte-compiled
```

More on this soon, but knowing that a file is byte-compiled Python code allows us to leverage various tools *specific to this file format* (for example, we can reconstruct a readable representation of the original python code using a python decompiler).

Returning to `OSX.WindTail`, we can again use the `file` utility to reveal that the malicious files (that masquerade as Office documents) are actually applications bundles, containing 64-bit Mach-O executables:

```
$ file Final_Presentation.app/Contents/MacOS/usrnode

usrnode: Mach-O 64-bit executable x86_64
```

Up Next

In this chapter, we introduced the concept of static analysis and highlighted how macOS's built-in `file` utility can effectively identify a file's true type. This, of course, is an important first analysis step as many static analysis tools are file type specific!

Next up, let's look at various file types one is likely to encounter while analyzing Mac malware. Some file types (e.g. disk images) are simply used to distribute malware and thus the goal is to extract the malicious contents for analysis. The actual malware comes in various other file formats, such as scripts and binaries.

For each file type, we'll briefly discuss its purpose, as well as highlight static analysis tools that can be used to analyze said file format.

References

1. “How to Reverse Malware on macOS Without Getting Infected”
   https://www.sentinelone.com/blog/how-to-reverse-macos-malware-part-one/

2. “Middle East Cyber-Espionage: Analyzing WindShift's implant: OSX.WindTail”
   https://objective-see.com/blog/blog_0x3B.html

3. “Mac Adware, à la Python”
   https://objective-see.com/blog/blog_0x3F.html

4. file utility
   x-man-page://file