Static Analysis

The technique of analyzing the suspect file without executing it

Common static analysis goals:

- Identifying the malware's target architecture
- Fingerprinting the malware
- Scanning the suspect binary with anti-virus engines
- Extracting strings, functions, and metadata associated with the file
- Identifying the obfuscation techniques used to thwart analysis
- Classifying and comparing the malware samples

1. Determining the file Type

Determining the file type will help determine the target OS ans architecture. File extensions are not the sole indicator of a file type, File signature can be used instead.

A File signature is a unique sequence of bytes that is written to the file's header. Different files have different signatures which can be used to identify the type of file. The Windows executables have a file signature of `MZ` or hexadecimal characters `4D 5A` in the first two bytes of the file.

1.1 Identifying the file using manual method

Open a file with a hex editor, and look for the file signature. (`xxd` on Linux and `HxD` on Windows)

1.2 Identifying File Type using Tools

Use file identification tools (`file` on Linux and `CFF Explorer` on Windows)

1.3 Determining the File using Python

The `python-magic` module can be used to determine the file type.

```python
import magic

m = magic.open(magic.MAGIC_NONE)
m.load()
print(m.file(r'log.exe'))
```

2. Fingerprinting the Malware

Fingerprinting involves generating the cryptographic hash values for the suspect binary based on it's file content. (Using `MD5, SHA1, SHA256`)

Useful because:

- Calculated based on file content, which remains the same (Unique identifier)
- During dynamic analysis, the file might copy and spread, hash helps to know if the analysis has to be performed on one file or on many more
- Used as an indicator to share with other security analysts
- Used to check if it's new (in *virusTotal*)

2.1 Generating Cryptographic hash using tools

On Linux: `md5sum, sha256sum sha1sum` On Windows: `HashMyFiles`

2.2 Determining cryptographic Hash with Python

Using the `hashlib` module

```python
import hashlib

content = open(r'log.exe', 'rb').read()
print(hashlib.md5(content).hexdigest()
```

3. Multiple Anti-virus scanning

Helps determine whether malicious code signatures exist for the suspect file. Allows you to gather more information on this file.

3.1 Scanning the Suspect Binary with VirusTotal

- Allows you to scan the virus sample with several anti-virus techniques.
- Allows you to search their database by *hash*, *URL*, *domain*, or *IP address*.
- VirusTotal Graph allows you to visualize relationships between files that you submitted and it's associated indicators (*URL*, *domain*, or *IP address*)

3.2 Querying hash values using VirusTotal Public API

It allows you to automate file submissions, retrieve file/URL scan reports, and retrieve domain/IP reports.

The alternatives to scripting is to use PE analysis tools, such as: *pestudio* or *PPEE*

There are a few risks to consider when scanning a binary with Anti-virus scanners:

- If the malware scanner does not detect the file as malware, this does not mean it is safe.
- The suspect malware may contain sensitive information specific to your organization.
- Attackers can use the scan feature, to search for their malware.

4. Extracting Strings

Extracting strings might give us an indication about the program functionality and indicators associated with a suspect binary.

4.1 String extraction using tools

- **Linux:** `Strings`
  - extracts strings from a given file
  - Can extract both *ASCII* and *Unicode* (`-el` option) strings from binary
- **Windows:** `pestudio`
  - Displays both *ASCII* and *Unicode* strings.

4.2 Decoding Obfuscated Strings Using FLOSS (FireEye Labs Obfuscated String Solver)

Designed to identify and extract obfuscated strings from malware automatically. It also extracts *stack strings*

5. Determining File Obfuscation

Malware authors obfuscate or armour their malware binary. Obfuscation is used by malware authors to protect the inner-workings of their program from security researchers, malware analysts and reverse engineers. They commonly use *Packers* and *Cryptors* to obfuscate their file.

5.1 Packers and Cryptors

- *Packer:* program that uses compression to obfuscate the executable's content. This obfuscated content is then stored within the structure of a new executable file; The result is a new executable file with obfuscated contents on the disk. Upon execution of the packed program, it executes a decompression routine, which extracts the original binary in memory during runtime and triggers the execution.
- *Cryptor:* Similar to *Packer* but instead of using compression, it uses encryption to obfuscate the executable's content.

5.2 Detecting File Obfuscation Using Exeinfo PE

- Use *Exeinfo PE* (Windows) to detect if the sample is packed.
- It uses more than 4500 signatures (Stored in `userdb.txt`) to detect various compilers, packers and cryptors utilized to build a program.
- It directs you on how to unpack the sample

6. Inspecting PE Header Information