A STEP BY STEP GUIDE for performing STATIC MALWARE ANALYSIS

Created by - Vaibhav Kamdi

Analysis Flow for Malware Analysis

- Setup a baseline analysis environment
- Triage to determine a starting point
- Static Analysis - Get a sense of where everything is before debugging.
- Dynamic Analysis - Determine behaviors that can’t be understood by static analysis.
- Manual Debugging - Stepping through the program to navigate to your goals.

Created by - Vaibhav Kamdi

Static Malware Analysis

- Lab Setup for malware analysis
- Searching for strings in a malware
- Examining the filetype
- Fingerprinting malware through hashes
- Signature-based detection mechanisms
- Extracting useful information from the PE Header

Created by - Vaibhav Kamdi

String:

```
$ strings filename.exe > filename.txt
$ strings filnemae | more
```

Filetype:

```
$ file <filename>
```

UPX:

Ultimate Packer for Executable (UPX) is an open-source packer that can reduce the file size of an executable drastically(better than Zip files) , and it is compatible with a large range of executable formats, like Windows DLLs, macOS apps, or Linux ELF.

Created by - Vaibhav Kamdi

- Calculating Hashes

```
$ md5sum <filename>
$ ssdeep
```

- AV scans and VirusTotal, AbuseIPDB
  - Detection
  - Behavior
  - Relations

- **Signature-based detection mechanisms**
  - Signatures are a way to identify if a particular file has a particular type of content. We can consider a signature as a pattern that might be found inside a file. This pattern is often a sequence of bytes in a file, with or without any context regarding where it is found. Security researchers often use signatures to identify patterns in a file, identify if a file malicious, and identify if a file contains the information of interest. Hashes are not the ideal tool to perform this task
  - **Yara rules**
  - **Proprietary Signatures** - Antivirus Scans
  - **Capa** - Capa is another open source tool created by Mandiant. This tool helps identify the capabilities found in a PE file. Capa read the files and tries to identify the behavior of the file based on signatures such as imports, strings, mutexes, and other artifacts presents in the file

- **PE File Header**
    - Linked Libraries, Imports/Exports
    - Identify Packed Executables
    - Sections
        - **.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.
    - $ pecheck <filename>
    - Tools
        - pe-tree
        - PEStudio
        - wxHexEditor

IMAGE_DOS_HEADER

- **IMAGE_DOS_HEADER** consists of the first 64 bytes of the PE file.
  - The first two bytes **4D 5A**, they translate MZ (**Mark Zbikowski**) characters in ASCII.
  - First entry in IMAGE_DOS_HEADER is e_magic value is 0x5a4d MZ
  - Last value in the IMAGE_DOS_HEADER is called e_lfanew - this denotes the address from where the IMAGE_NT_HEADERS start

DOS_STUB

- It is a small piece of code that only runs if the PE file is incompatible with the system it is being run on.

IMAGE_NT_HEADER

- The starting address of IMAGE_NT_HEADERS is found in e_lfanew from the IMAGE_DOS_HEADER.(offset)
- The first 4 bytes of the NT_HEADERS consist of the Signature. We can see this as the bytes 50 45 00 00 in Hex.
- FILE_HEADER contains some vital information
  - Machine
  - NumberOfSections
  - TimeDateStamp
  - PointerToSymbolTable and NumberOfSymbols
  - SizeOfOptionalHeader
  - Characteristics

OPTIONAL_HEADER

- Critical field of OPTIONAL_HEADER:
  - **Magic :** It tells PE file is a 32-bit or 64-bit application. If the value is 0x010B, it denotes a 32-bit application; if the value is 0x020B, it represents a 64-bit application
  - **AddressOfEntryPoint:** This field is significant from a malware analysis/reverse-engineering point of view. This is the address from where Windows will begin execution.
  - **BaseOfCode** and **BaseOfData:** These are the addresses of the code and data sections, respectively, relatively to ImageBase.
  - **ImageBase:** The ImageBase is the preferred loading of the PE file in memory. Generally, the ImageBase for .exe files is 0x00400000.

OPTIONAL_HEADER (Continued..)

- Critical field of OPTIONAL_HEADER:
    - **Subsystem:** This represents the Subsystem required to run the image. This Subsystem can be Windows Native, GUI (Graphical User Interface), CUI (Commandline User Interface), or other Subsystem. Subsystem 0x0002 representing Windows GUI.
    - **DataDirectory:** The DataDirectory is a structure that contains import and export information of the PE file (called Import Address Table and Export Address Table)

- **Packing and identifying packed executables**
  - Unconventional section names
  - **EXECUTE** permission for multiple sections
  - High Entropy, approaching 8, for some sections
  - A significant difference between SizeOfRawData and Misc_VirtualSize of some PE sections
  - Very few important functions

Created by - Vaibhav Kamdi