libpe Documentation

Release 0.1.1

Fernando Merces

December 10, 2015

Contents

`libpe` is a malware analysis library that allows you to decode a stream of bytes from a PE file (`.exe`, `.dll`, `.fon` etc), inspect various bits of information about them, and translate them to a human-friendly format. `libpe` is free software, distributed under the terms of the GNU Lesser General Public License.

CHAPTER 1

Building and installing `libpe`

libpe is developed for unix-like environments, so the basic steps towards building and installing it are the usual:

```
make
make install
```

The installation routines copy the necessary library files to the appropriate locations in your system. Depending on your choice of installation location, though, you may need to have root privileges to run the second command on unix-like environments.

CHAPTER 2

Using `libpe` within your program - a quick example

The following is an example of a program (`exe-check.c`) that incorporates `libpe` and uses its API to generate information about a Windows executable (`suspicious.exe`). It assumes you have all `libpe`'s headers in the same directory as the main source file.

```c
#include <stdio.h>
#include <stdlib.h>
#include "pe.h"

int main(void) {
    // Open binary file for parsing
    pe_ctx_t ctx;
    pe_err_e err = pe_load_file(&ctx, "suspicious.exe");
    if (err != LIBPE_E_OK) {
        pe_error_print(stderr, err);
        return EXIT_FAILURE;
    }

    // Parse binary file
    err = pe_parse(&ctx);
    if (err != LIBPE_E_OK) {
        pe_error_print(stderr, err);
        return EXIT_FAILURE;
    }

    if (!pe_is_pe(&ctx)) {
        return EXIT_FAILURE;
    }

    // Get COFF header information and output it
    IMAGE_COFF_HEADER *coff = pe_coff(&ctx);
    printf("Machine: %x\n", coff->Machine);

    return 0;
}
```

To compile the program with gcc:

```
gcc exe-check.c -o exe-check -std=c99 -L. -lpe
```

This example should give you an idea of how `libpe` can be used in your programs. You first load the executable (*load* here doesn't mean *load in memory*, but simply *open for reading*), parse its contents and display some information contained in its COFF header (`coff->machine` represents the target machine `suspicious.exe` was compiled for). Note that error checks are performed for each of those steps.

The following sections describe `libpe`'s API in more detail.

CHAPTER 3

libpe's API

3.1 Context (pe_ctx_t)

As you saw in the *previous example*, you need to define a variable of type `pe_ctx_t` in order to use `libpe`'s API (`pe_load_file`, `pe_parse` etc.). Think of `pe_ctx_t` as a general-purpose placeholder that will contain the main information about the PE file to be analyzed. Here's its declaration:

```
typedef struct {
    FILE *stream;
    char *path;
    void *map_addr;
    off_t map_size;
    uintptr_t map_end;
    pe_file_t pe;
} pe_ctx_t;
```

`stream` is a file descriptor that points to the PE file once it's opened. `path` is a string representing the absolute path to the PE file in the filesystem. `map_addr`, `map_size` and `map_end` are used in the PE file's address mapping (see mmap(2)). Finally, `pe_file_t` is another type internally declared by `libpe` (more on it *below*).

3.2 PE file (pe_file_t)

```
typedef struct {
        IMAGE_DOS_HEADER *dos_hdr;
        uint32_t signature;
        IMAGE_COFF_HEADER *coff_hdr;
        void *optional_hdr_ptr;
        IMAGE_OPTIONAL_HEADER optional_hdr;
        uint32_t num_directories;
        void *directories_ptr;
        IMAGE_DATA_DIRECTORY **directories;
        uint16_t num_sections;
        void *sections_ptr;
        IMAGE_SECTION_HEADER **sections;
        uint64_t entrypoint;
        uint64_t imagebase;
} pe_file_t;
```