April 8, 2014

Examining Unknown Binary Formats

This post is about to discuss the methods for examining unknown binary formats that can be either a file, file fragment, or memory dump.

Before discussing the methods I'm describing few scenarios when examination of an unknown format is appropriate.

Imagine you deal with an application that handles a certain file type that you want to fuzz. You think to carry out some dumb fuzzing with a bit of improvement. Before doing so, you may be examining the format to create an approximate map of the layout. So you'll get an idea what parts of the files are worth fuzzing, and what fuzzing method is reasonable to apply for each part.

In other scenario you might have the binary file but don't have the program that parses it. You want to know as much as possible of the format of the binary file to understand it's layout.

If the application that reads the file format is available you can use debugger to watch how the data is parsed. This scenario is not discussed here.

If the application that writes the format is available you can try the following idea. You may produce output file using the application. This can be done by save, export, convert options available in the application. Next time when producing output you change something minor in the settings that may produce a similar output file. Comparing the two output files you may see what changed.

Entropy analysis is very useful to locate compressed, encrypted, and other way encoded data. Higher entropy can indicate encoding of some kind. Lower entropy is likely anything else including text, code, header, data structures. Redundancy analysis is analogue to entropy analysis; the lower the redundancy the most likely the data is encoded.

Encoded data could be anything, even multimedia data. The compressed streams can have headers and/or magic bytes identify the compression type.

Character distribution of the file can tell us a lot. Creating a byte frequency map is very straightforward by using modern programming languages. That can tell us what are the most and less frequent bytes. We can easily know what are the bytes that are not present at all.

Strings can be discovered even with popular tools like a hex-editor. Most common encodings are ASCII and Unicode. If there is no terminating zero the length of the string is likely stored somewhere in the binary. It's often the preceding byte(s) of the first letter of the string.

Consecutive patterns, byte sequences are seen to be used for padding, for alignment, or to fill slack space.

Random-looking printable characters can indicate some kind of encoding of any data in plain text.

Scattered bytes, scattered zeros, scattered 0FFh bytes can indicate sequence of encoded integers. Integers can be offsets and lengths. Scattered zeros might indicate text in Unicode format.

It could be useful to analyze the density of zeros, printable characters, or of other patterns. This could be applied on the whole file or on a particular region of the file.

Consecutive values, integers might indicate an array of pointers. It might be useful to know if the values increasing, decreasing, or random values.

Also, good to know in what endianness the integers stored.

x86 code can be be detected by running disassembler on the binary. If you see a sequence of meaningful instructions that might be code-area.

There is a simpler way to look for x86 code though. You write a small program in some high level language that searchers for E8 (CALL) / E9 (JMP) patterns and calculates the absolute offset where the instruction jumps. If there is an absolute offset referenced from different places that might be an entry point of a real function. The more functions are identified the better the chance you have found code.

If you know what native code to look for you can search for a sequence of common instructions, like bytes at function entry point.

Meaningful text fragment in high-entropy area might indicate run-length encoding which is also known as RLE compression.

There is data format that looks like this. It consists of a sequence of structures, or chunks. The size of each structure is encoded sometimes as a first value in the structure. It's commonly seen that a sequence of compressed data is stored like that.

If it's known the binary is associated with certain time stamp or version number those constants might worth searching for.

Some methods described here can be combined with xor-search, and with other simple decoding techniques to discover the structure of the file.