Ramy NAIEM

Milestone 1



Sample 1

(Date of the file compilation, Language, developer, architecture, sections, properties,memory locations, memory size)

File type: ELF 64



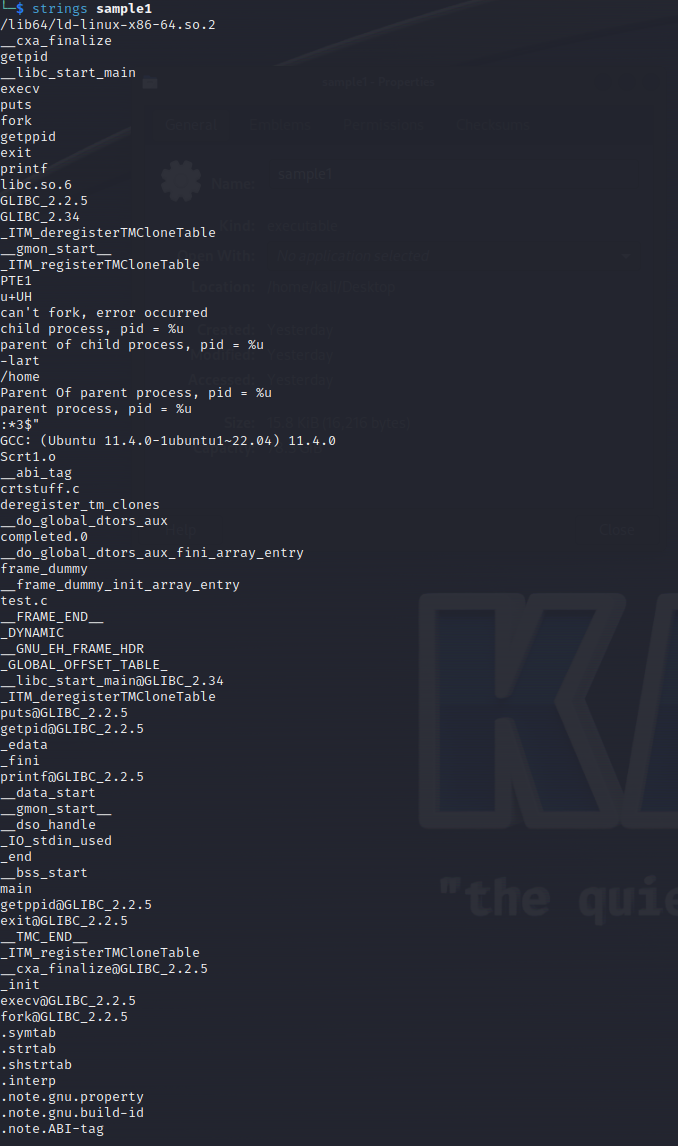
File size: 15.84 KB (16216 bytes)



Date: Sunday 29 October 2023, 00.21.54



Language: it seems like this program was created using the GNU Compiler Collection (GCC), and it's likely written in C or C++. Some of the strings you found, such as "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0," strongly indicate that it was compiled with GCC. Additionally, common C library functions like printf, getpid, fork, and exit are present, suggesting that the code is likely in C or C++.



Signature:

Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00



Developer:

Architecture: "Advanced Micro Devices X86-64." This architecture is commonly known as "x86-64" or "AMD64."

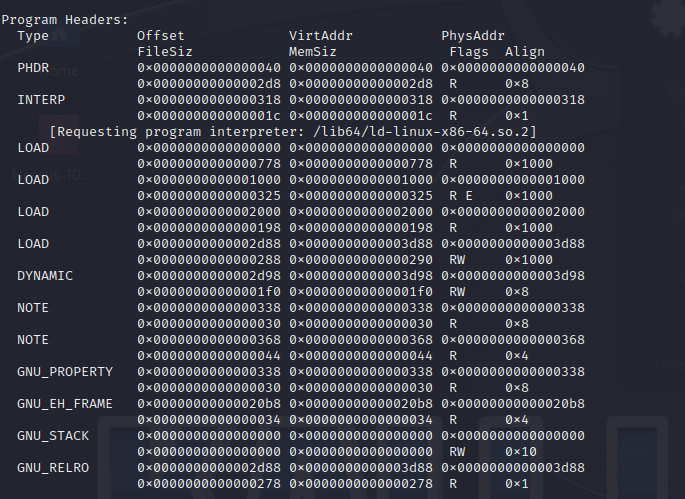


Sections: 31 sections



Properties: x86 feature: IBT, SHSTKx86 ISA needed: x86-64-baseline

Memory locations



Memory size

PHDR: MemSiz is 0x2d8 (728 bytes).

INTERP: MemSiz is 0x1c (28 bytes).

LOAD: There are multiple LOAD program headers, each with its own MemSiz:

The first LOAD header has MemSiz of 0x778 (1912 bytes).

The other LOAD headers have MemSiz of 0x325 (805 bytes).

DYNAMIC: MemSiz is 0x1f0 (496 bytes).

NOTE: There are several NOTE headers in the program headers section, and each has its own MemSiz:

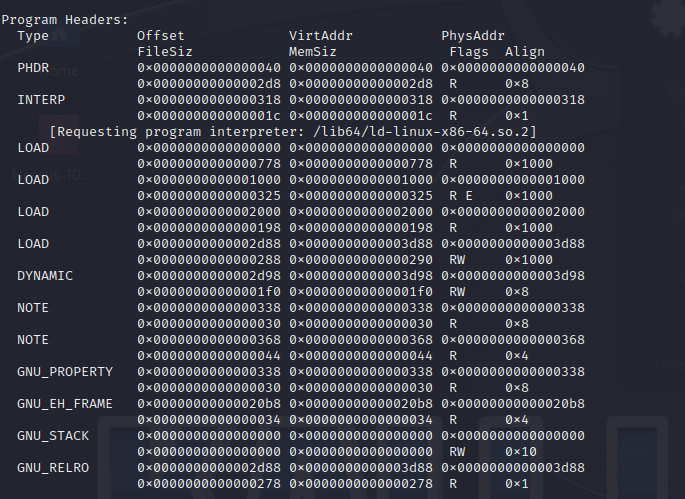
The size for the NOTE headers may vary, as indicated in the section headers. You can check the respective MemSiz for each NOTE header in the section headers.

GNU\_PROPERTY: MemSiz is 0x30 (48 bytes).

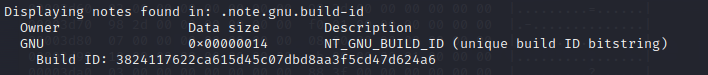
GNU\_EH\_FRAME: MemSiz is 0x34 (52 bytes).

GNU\_STACK: MemSiz is 0x0 (0 bytes).

GNU\_RELRO: MemSiz is 0x278 (632 bytes).



Owner: GNU



OS: OS: Linux, ABI: 3.2.0



Data: 2's complement, little endian

More information:

Relocation Information: Relocation information is usually present in ELF files and tells the linker how to change addresses when merging many object files together. Understanding how the program interacts with other files and libraries during linking can be gained by examining the relocation entries.

Symbol Tables: Symbol tables are a common feature of ELF files and are essential for debugging and dynamic linking. You can learn about the functions, variables, and symbols used in the program by looking through the symbol tables. When debugging or reverse engineering software, this is really helpful.

Sections that are Dynamic: These sections provide details on the program's runtime execution and dynamic linking dependencies. Finding out which shared libraries the program depends on and what runtime relocations need to be done can be accomplished by looking at the dynamic sections.

**Sample 2**

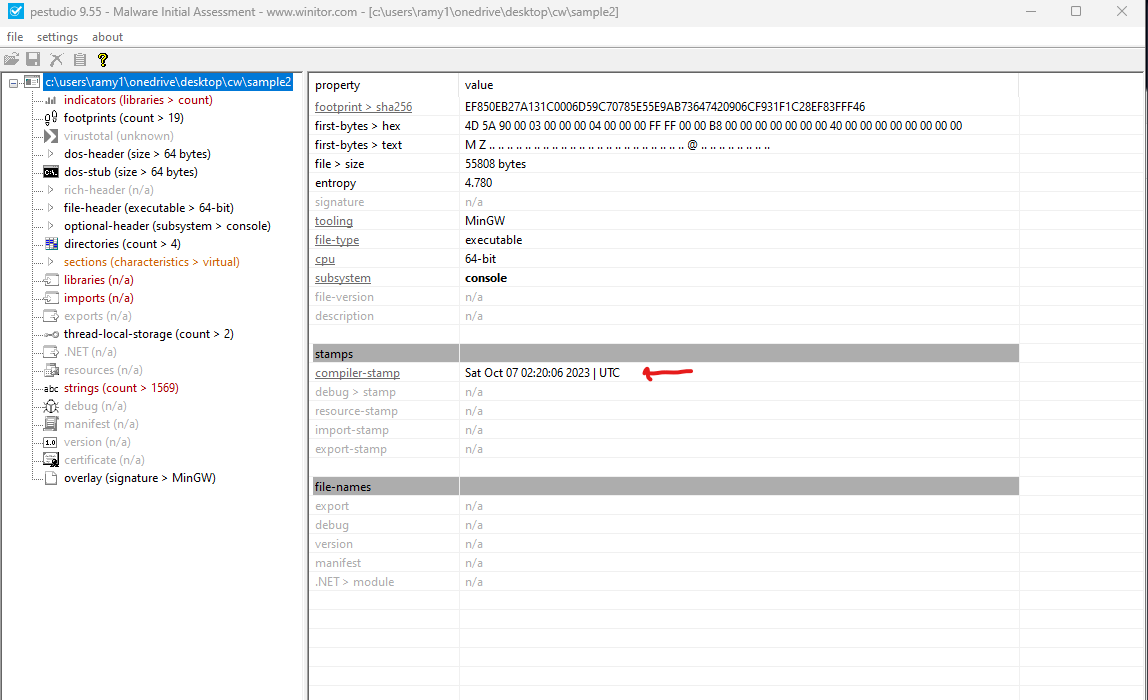
(Date of the file compilation, Language, developer, architecture, sections, properties,memory locations, memory size)

File type: Portable Executable 64

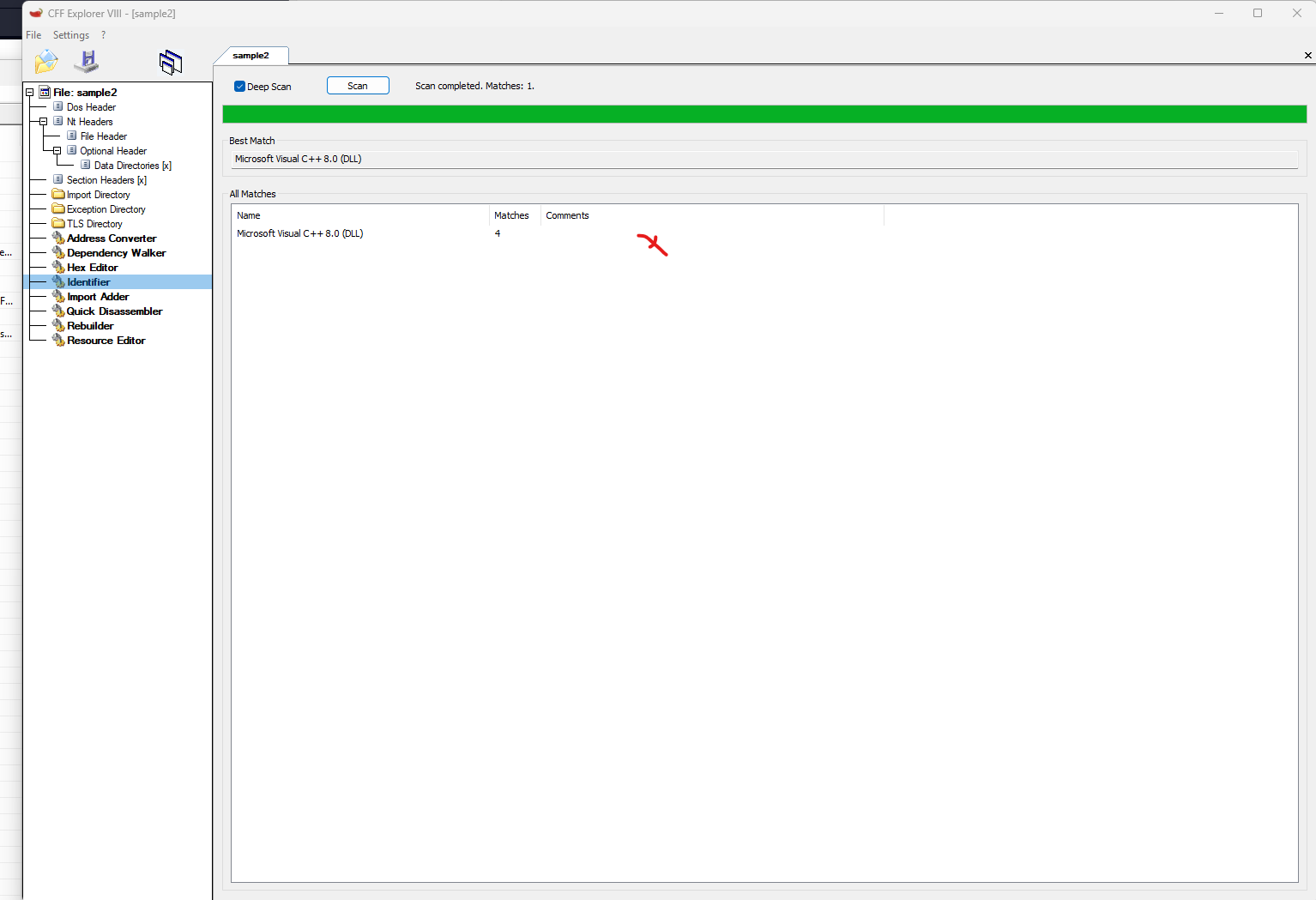
File size: 54.50 KB (55808 bytes)



Date: compiler-stamp,Sat Oct 07 02:20:06 2023 | UTC



Language: Best Match: Microsoft Visual C++ 8.0 (DLL)



All Matches:

Signature: Microsoft Visual C++ 8.0 (DLL)

Matches: 4

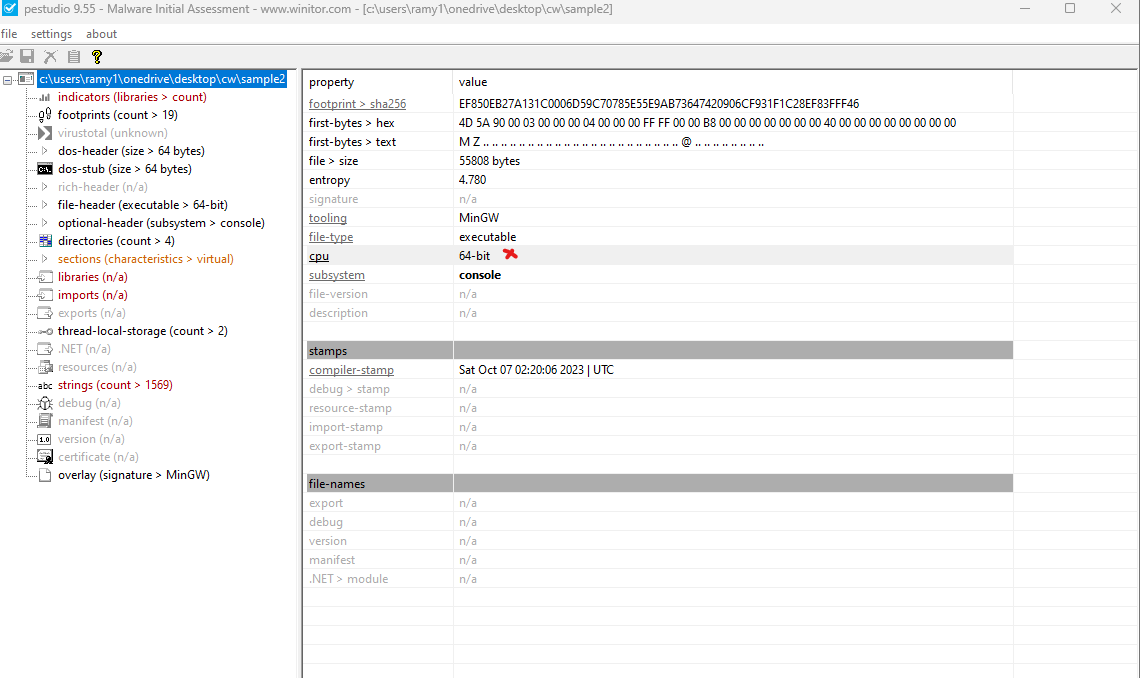
Developer:



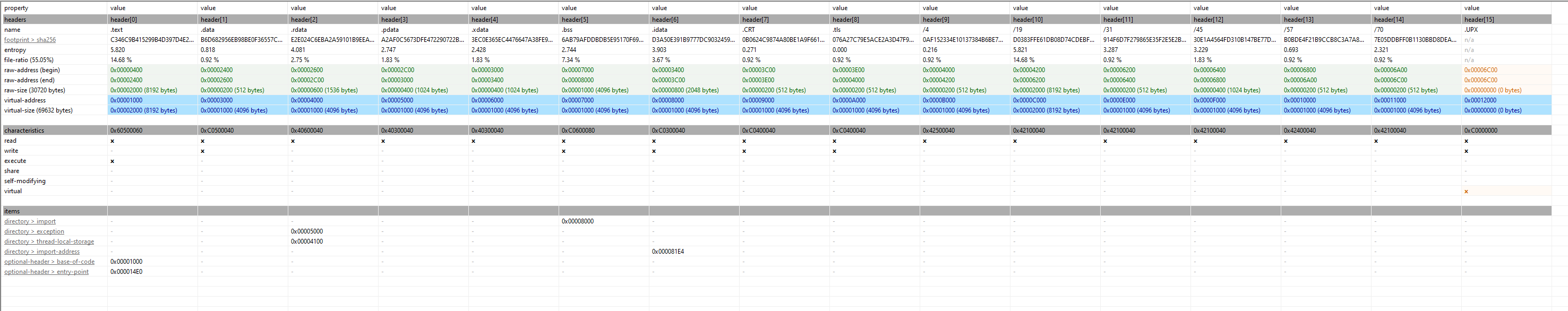
<https://en.wikipedia.org/wiki/DOS_MZ_executable> :



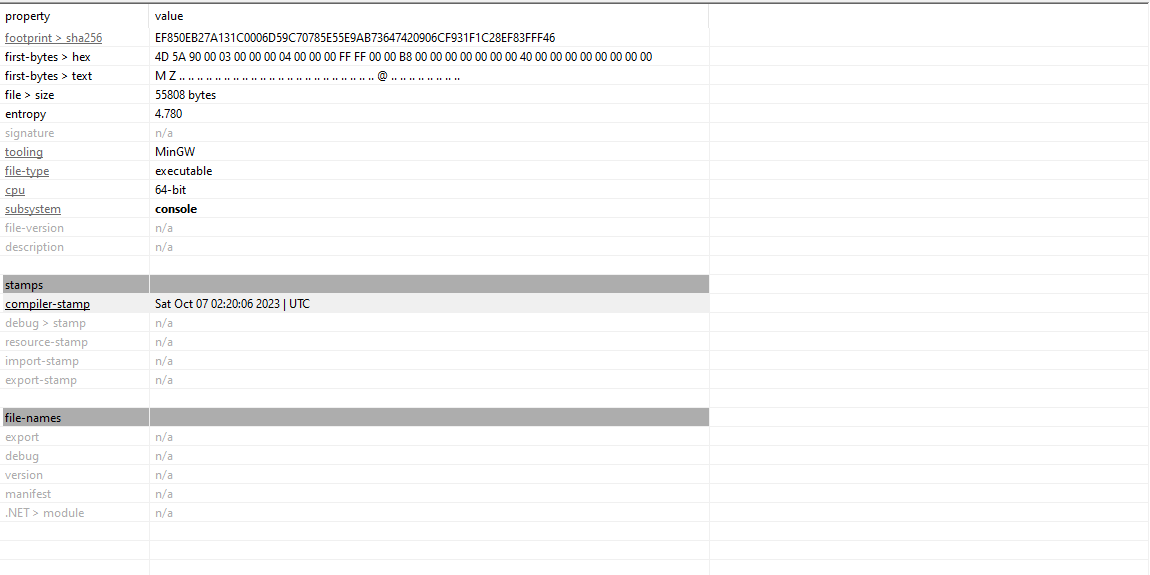
Architecture: machine,0x8664,Amd64, cpu,64-bit



Sections



Properties



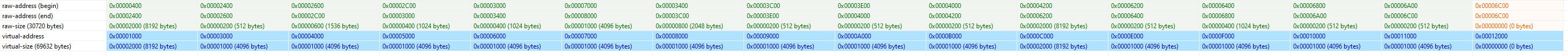
Memory locations

Raw Address (Begin): This is the starting address of the section within the file on disk. It represents the offset from the beginning of the file to the start of the section. It is often used when referring to the file layout.

Raw Address (End): This is the ending address of the section within the file on disk. It represents the offset from the beginning of the file to the end of the section.

Virtual Address: This is the address at which the section will be loaded into memory when the program is executed. It represents the address in the program's memory space.

1. Header 0  
     
   raw-address (begin) 0x00000400  
   raw-address (end) 0x00002400  
   virtual address 0x00001000
2. Header 1  
     
   raw-address (begin) 0x00002400  
   raw-address (end) 0x00002600  
   virtual address 0x00003000
3. Header 2  
     
   raw-address (begin) 0x00002600  
   raw-address (end) 0x00002C00  
   virtual address 0x00004000
4. Header 3  
     
   raw-address (begin) 0x00002C00  
   raw-address (end) 0x00003000  
   virtual address 0x00005000
5. Header 4  
     
   raw-address (begin) 0x00003000  
   raw-address (end) 0x00003400  
   virtual address 0x00006000
6. Header 5  
     
   raw-address (begin) 0x00007000  
   raw-address (end) 0x00008000  
   virtual address 0x00007000
7. Header 6  
     
   raw-address (begin) 0x00003400  
   raw-address (end) 0x00003C00  
   virtual address 0x00008000
8. Header 7  
     
   raw-address (begin) 0x00003C00  
   raw-address (end) 0x00003E00  
   virtual address 0x00009000
9. Header 8  
     
   raw-address (begin) 0x00003E00  
   raw-address (end) 0x00004000  
   virtual address 0x0000A000
10. Header 9  
      
    raw-address (begin) 0x00004000  
    raw-address (end) 0x00004200  
    virtual address 0x0000B000
11. Header 10  
      
    raw-address (begin) 0x00004200  
    raw-address (end) 0x00006200  
    virtual address 0x0000C000
12. Header 11  
      
    raw-address (begin) 0x00006200  
    raw-address (end) 0x00006400  
    virtual address 0x0000E000
13. Header 12  
      
    raw-address (begin) 0x00006400  
    raw-address (end) 0x00006800  
    virtual address 0x0000F000
14. Header 13  
      
    raw-address (begin) 0x00006800  
    raw-address (end) 0x00006A00  
    virtual address 0x00010000
15. Header 14  
      
    raw-address (begin) 0x00006A00  
    raw-address (end) 0x00006C00  
    virtual address 0x00011000
16. Header 15  
      
    raw-address (begin) 0x00006C00  
    raw-address (end) 0x00006C00  
    virtual address 0x00012000



Memory size

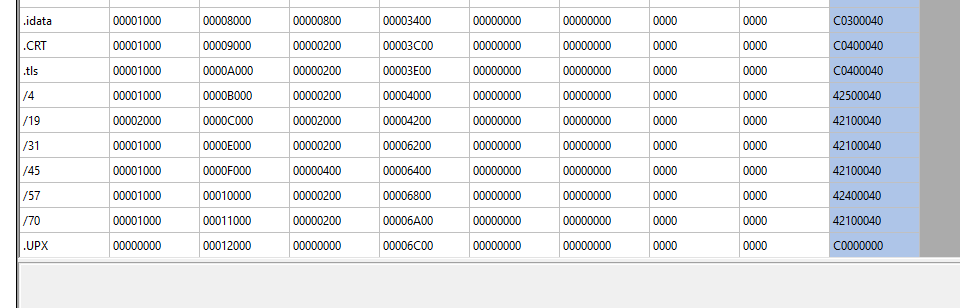
Virtual Size (VirtSize): This indicates the size of the section when loaded into memory. It represents the amount of memory space that the section will occupy when the executable is run.

Raw Size (RawSize): This is the size of the section within the file itself. It represents the amount of space the section occupies in the file on disk.

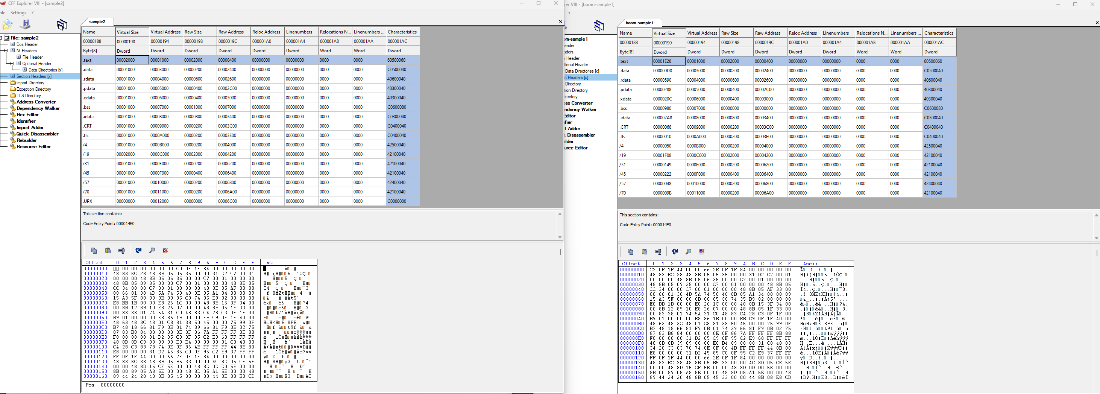
Corrupted file:

File 2 is the corrupted file because I have compared file 2 to the one that was provided in the lecture, and I have seen that it has added .upx to the sample 2 and the .bss was different everything else was the same as the boom-sample1. So, I have tried to rebuilder in cff explorer I have used rebuild PE header and update Checksum. I have unpacked the file in Linux to see if this lifts the corruption, but it told me that the file was not packaged with upx, so I had to find another solution. The way I fixed the corruption I removed the upx header using cff

Deffinition of .bss : When referring to an executable and linkable format (ELF) file, the ".bss" portion is crucial. "bss" is an acronym for "Block Started by Symbol." Uninitialized data, in particular variables declared within the program without initial values, are stored in this specific section. The information about the size and memory location of these uninitialized variables is what sets the.bss section different.



We can also see that .text is different



I have also provided the fixed sample 2 file.