

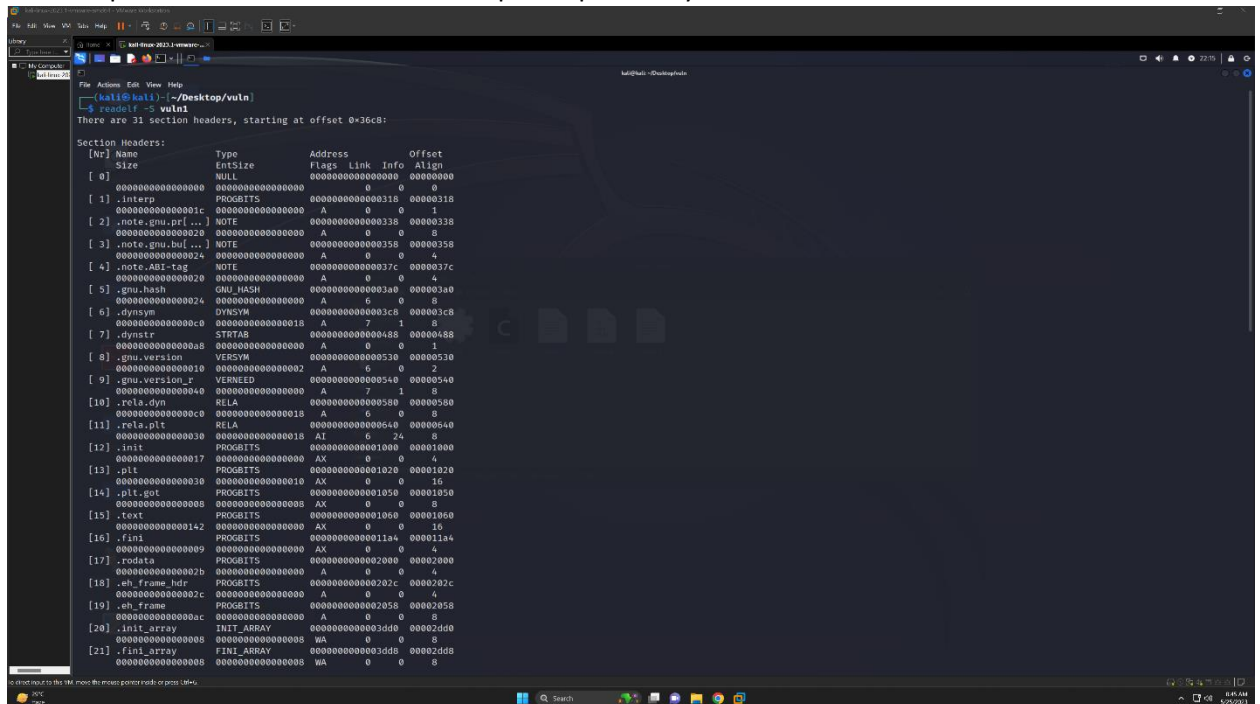
Name: Archit Benipal

Subject: Software Supply Chain Security (CY 653)

Assignment 3: Derive its sections information from an executable

1. Use 'readelf -S vuln1' to get all the sections that have been integrated into vuln1. Ask ChatGPT for the usage of these sections. Write a report about the sections in a ELF file.

➔ The **readelf** utility displays information about one or more ELF format object files. This command is used in Linux systems to display the sections of an ELF (Executable and Linkable Format) file, which is often used for executables, object code, shared libraries, and core dumps. The **-S** or **--section-headers** option specifically lists the section headers.



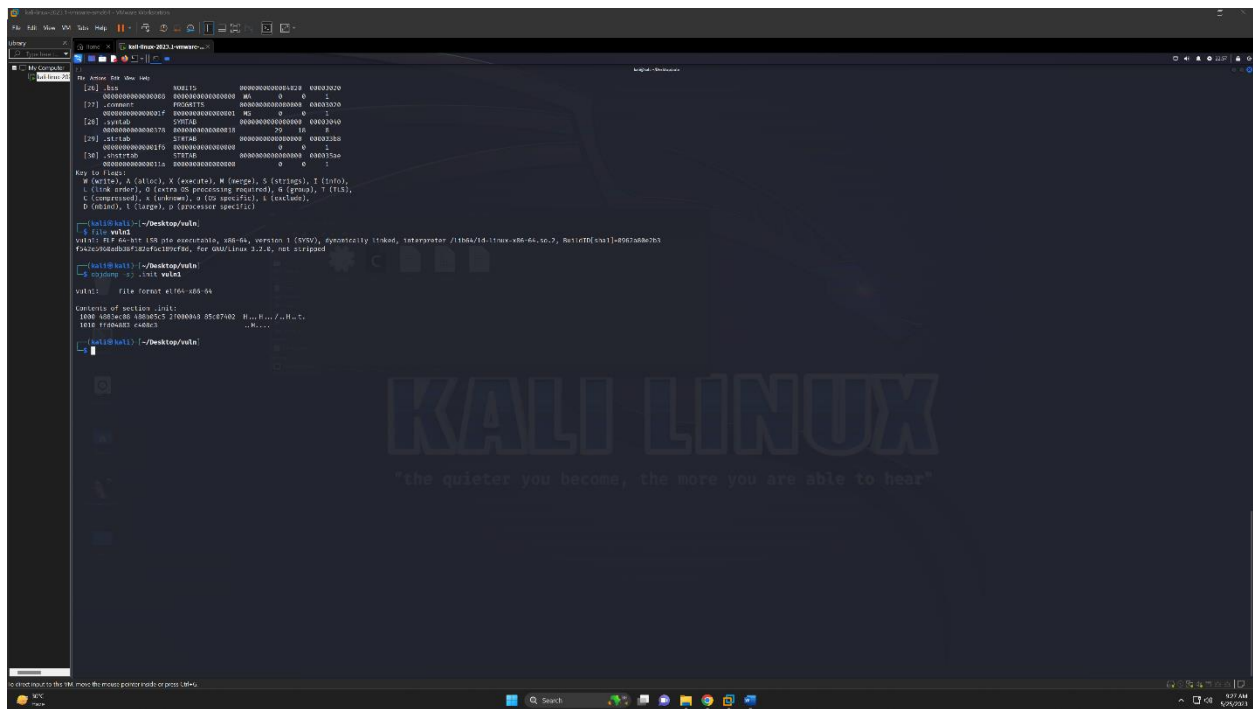
```
File Actions Edit View Help
[~(kali@kali) - ~/Desktop/vuln1]
$ readelf -S vuln1
There are 31 section headers, starting at offset 0x36c8:

Section Headers:
[Nr] Name           Type             Address           Offset
----
[ 0] .null              NULL             0000000000000000 00000000
[ 1] .interp            PROGBITS         0000000000000318 00000318
[ 2] .note.gnu.pr[...] NOTE             0000000000000338 00000338
[ 3] .note.gnu.bu[...] NOTE             0000000000000358 00000358
[ 4] .note.ABI-tag      NOTE             000000000000037c 0000037c
[ 5] .gnu.hash           GNU_HASH         00000000000003a0 000003a0
[ 6] .dynsym             DYNAMIC          00000000000003c8 000003c8
[ 7] .dynamic            DYNAMIC          0000000000000400 00000400
[ 8] .gnu.version        VERSYM           0000000000000530 00000530
[ 9] .gnu.version_r      VERNEED          0000000000000540 00000540
[10] .rela.dyn            RELA             0000000000000580 00000580
[11] .rela.plt            RELA             0000000000000640 00000640
[12] .init               PROGBITS         0000000000001000 00001000
[13] .plt                PROGBITS         0000000000001020 00001020
[14] .plt.got            PROGBITS         0000000000001050 00001050
[15] .text               PROGBITS         0000000000001060 00001060
[16] .fini               PROGBITS         0000000000001144 00001144
[17] .rodata             PROGBITS         0000000000002000 00002000
[18] .eh_frame_hdr       PROGBITS         000000000000202c 0000202c
[19] .eh_frame           PROGBITS         0000000000002058 00002058
[20] .init_array          INIT_ARRAY       0000000000003d00 00003d00
[21] .fini_array          FINI_ARRAY       0000000000003d08 00003d08
```

Upon running the command I got 31 section headers.

### Description of each section headers:

- i. **.interp:** This section holds the path name of a program interpreter. If the file has a loadable segment that includes relocation, the sections' attributes will include the SHF\_ALLOC bit; otherwise, that bit will be off. If the file has a loadable segment that includes relocation, then the sections' attributes (sh\_addr) will give the virtual address of that interpreter.
- ii. **.note:** This section holds information in the format of the Note Section and **.note.gnu.property**, **.note.gnu.build-id**, **.note.ABI-tag** these sections hold note information. The note section is often used for vendor-specific information like ABI versions or the GNU Build ID.  
**Note Section:** Sometimes a vendor or system builder needs to mark an object file with special information that other programs will check for conformance, compatibility, etc. Sections of type SHT\_NOTE and program header elements of type PT\_NOTE can be used for this purpose. The note information in sections and program header elements holds a variable amount of entries. In 64-bit objects (files with e\_ident[EI\_CLASS] equal to ELFCLASS64), each entry is an array of 8-byte words in the format of the target processor. In 32-bit objects (files with e\_ident[EI\_CLASS] equal to ELFCLASS32), each entry is an array of 4-byte words in the format of the target processor. Labels appear below to help explain note information organization, but they are not part of the specification.
- iii. **.gnu.hash:** The GNU hash section is an improved version of SysV-style hashing which improves symbol lookup time. It's used by the dynamic linker to look up symbols in the binary.
- iv. **.dynsym:** This section holds the dynamic linking symbol table. If the binary is dynamically linked, all the symbols that are needed for dynamic linking are in this table.
- v. **.dynstr:** This section holds strings needed for dynamic linking, most commonly the strings that represent the names associated with symbol table entries.
- vi. **.gnu.version:** The version symbol table. This section holds version information for the symbols defined in the dynamic symbol table.
- vii. **.gnu.version\_r:** This section holds the version dependency information. That is, it tells what versions of what shared libraries the binary depends on.
- viii. **.rela.dyn:** This section holds RELA type relocation information for all sections of a shared library except the PLT.
- ix. **.rela.plt:** This section holds RELA type relocation information for the PLT section of a shared library or dynamically linked application.
- x. **.init:** This section holds executable instructions that contribute to the process initialization code. When a program starts to run, the system arranges to execute the code in this section before calling the main program entry point.



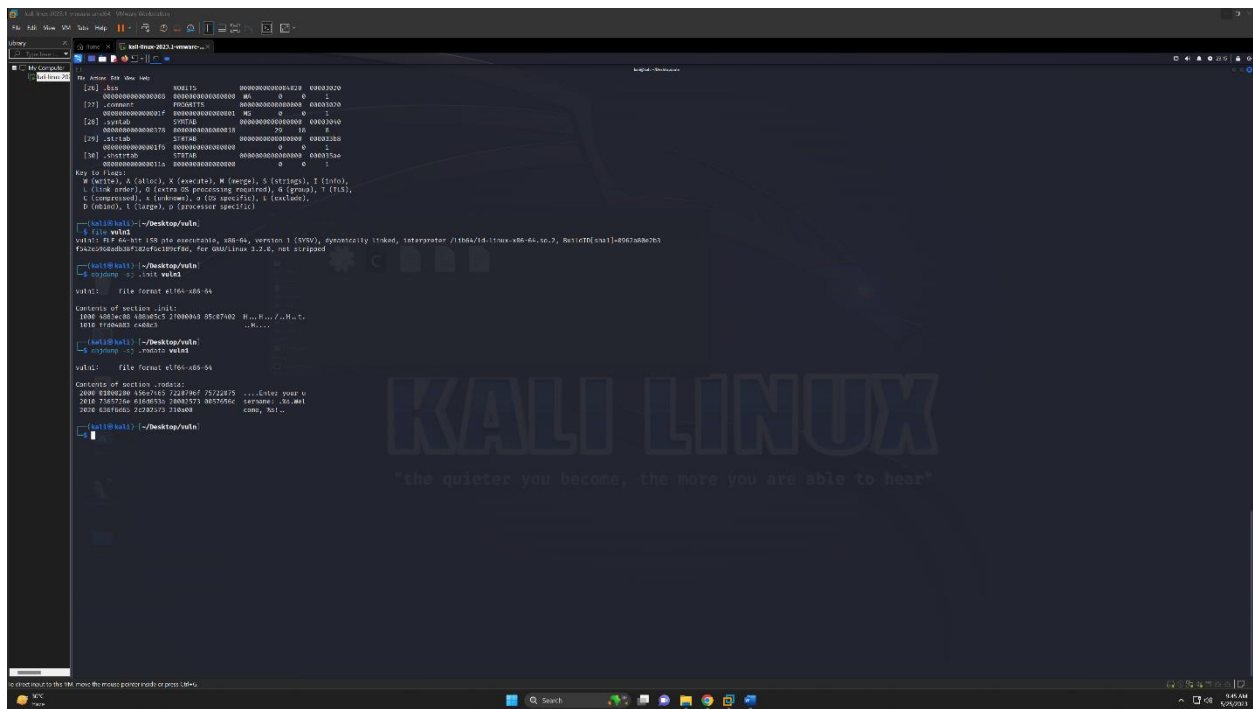
Here I have used objdump to view the contents of the .init section for vuln1

- xi. **.plt**: This section holds the procedure linkage table.
- xii. **.plt.got**: The difference between .plt and .plt.got is that .plt uses lazy binding and .plt.got uses non-lazy binding.

The difference between `.plt` and `.plt.got` is that `.plt` uses lazy binding and `.plt.got` uses non-lazy binding.

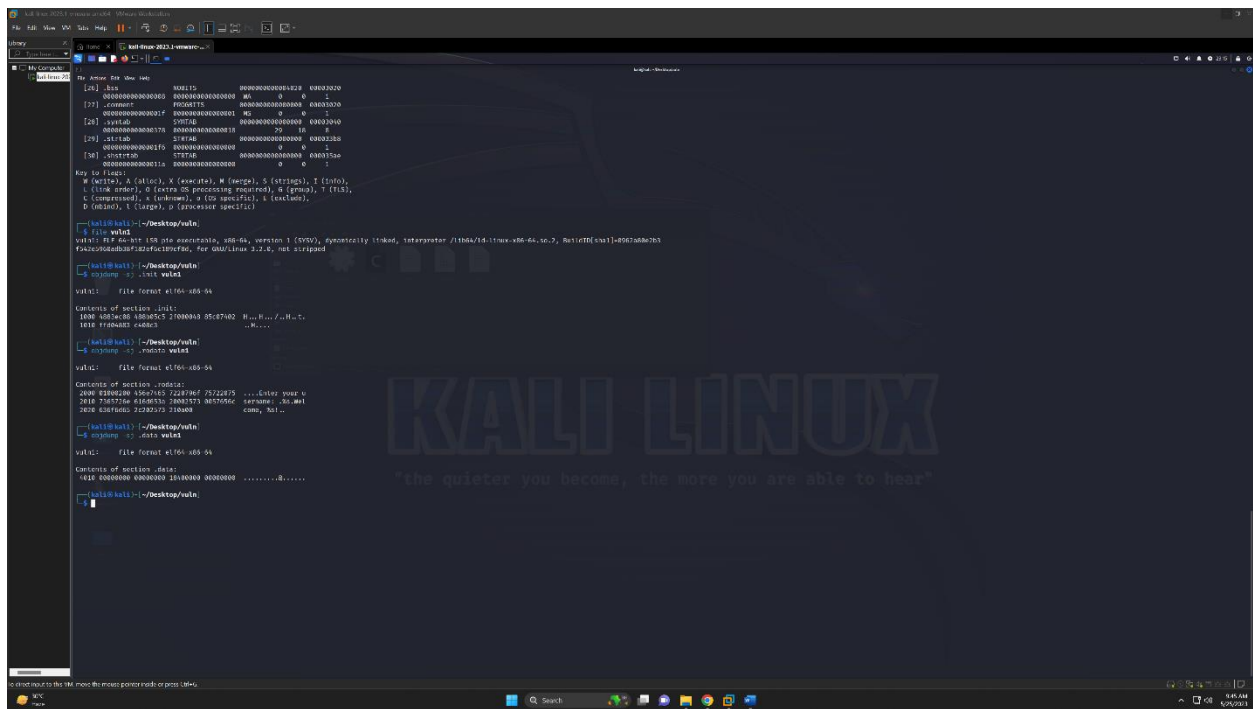
Lazy binding is possible when all uses of a function are simple function calls. However, if anything requires the address of the function, then non-lazy binding must be used, since binding can only occur when the function is called, and we may need to know the address before the first call. Note that when obtaining the address, the GOT entry is accessed directly; only the function calls go via `.plt` and `.plt.got`. If the `-fno-plt` compiler option is used, then neither `.plt` nor `.plt.got` are emitted, and function calls also directly access the GOT entry.

- xiii. **.text:** This section holds the ``text," or executable instructions, of a program.
- xiv. **.fini:** This section holds executable instructions that contribute to the process termination code. That is, when a program exits normally, the system arranges to execute the code in this section.
- xv. **.rodata:** This section holds read-only data, such as string literals and constants.



Here I have used objdump to generate the contents of the .rodata

- xvi. **.eh\_frame\_hdr:** It is used in runtime to access the eh\_frame. That means, it contains the pointer and binary search table to efficiently retrieve the information from eh\_frame.
- xvii. **.eh\_frame:** it contains exception unwinding and source language information. Each entry in this section is represented by single CFI (call frame information).
- xviii. **.init\_array:** This section holds an array of function pointers that contributes to a single initialization array for the executable or shared object containing the section.
- xix. **.fini\_array:** This section holds an array of function pointers that contributes to a single termination array for the executable or shared object containing the section.
- xx. **.dynamic:** This section holds dynamic linking information. The section's attributes will include the SHF\_ALLOC bit. Whether the SHF\_WRITE bit is set is processor specific.
- xxi. **.got:** This section holds the global offset table.
- xxii. **.got.plt:** The .got.plt section contains entries that are used by the dynamic linker to resolve function addresses at runtime. Every time a dynamic function is called for the first time, control is transferred to the .plt section which then uses the .got.plt to find the function address. If the function address is not found, the dynamic linker is invoked to find the function and populate the .got.plt for subsequent calls.
- xxiii. **.data:** This section holds initialized global and static variables.



Used objdump to view the contents of the .data

- xxiv. **.bss:** This section holds uninitialized data that contribute to the program's memory image. By definition, the system initializes the data with zeros when the program begins to run. The section occupies no file space, as indicated by the section type, SHT\_NOBITS.
- xxv. **.comment:** This section holds version control information.
- xxvi. **.symtab:** This section holds a symbol table. If the file has a loadable segment that includes the symbol table, the section's attributes will include the SHF\_ALLOC bit; otherwise, that bit will be off.
- xxvii. **.strtab:** This section holds strings, most commonly the strings that represent the names associated with symbol table entries. If the file has a loadable segment that includes the symbol string table, the section's attributes will include the SHF\_ALLOC bit; otherwise, that bit will be off.
- xxviii. **.shstrtab:** This section holds strings used to index the section header table.

### References:

- <https://refspecs.linuxbase.org/elf/gabi4+/ch4.sheader.html>
- [https://refspecs.linuxbase.org/elf/gabi4+/ch5.pheader.html#note\\_section](https://refspecs.linuxbase.org/elf/gabi4+/ch5.pheader.html#note_section)
- <https://stackoverflow.com/questions/58076539/plt-plt-got-what-is-different>
- <https://refspecs.linuxbase.org/elf/gabi4+/ch4.symtab.html>