Part B, Binary Analysis Report:

- My C Code:
  - The C code I have chosen to use is a basic password checker. It takes a single command-line argument and compares it to a global password. If the global password matches, access would be granted and a global counter would also increase. If the password is incorrect, there is an error message. My entire code can be seen in *Figure 1*.
  - I thought this C code would be good for this analysis because it does touch upon key ELF sections such as:
    - `.text` - all the executable code in the `main`, `check_password`, and `show_usage` functions
    - `.data` - the initialized global variables like counter and the string constants like `ACCESS_MESSAGE` and `GLOBAL_PASSWORD`
  - I also thought the use of multiple functions and standard library functions would work well when analyzing the symbol table and `PLT`
  - Overall, the code seemed simple enough while still holding a good number of functions and variables that make analysis possible. It being a password checker also felt in theme with the class.
- ELF Header: *Figure 2*
  - What is the ELF file type and target architecture?
    - The ELF file type can be seen as `DYN`, or shared object file. This usually occurs when a program is compiled with Position Independent Executable (`PIE`) enabled by default.
    - The architecture is listed as `Advanced Micro Devices X86-64`. This means it is a `64-bit x86` architecture, which is seen as common for current Linux systems.
  - What is the entry point of the binary?
    - The entry point address is listed as `0x10e0`, which is where the runtime begins executing the program.
- ELF Sections: *Figure 3*
  - Which section contains the executable code?
    - The `.text` section seen on line 16 contains the program's executable code. In my C code, it begins at `0x10e0` and contains compiled code for `main`, `check_password`, and `show_usage`
  - How do `.data` and `.bss` differ?
    - The `.data` section on line 25 contains initialized global variables, like counter, and the hardcoded string macros. This section allocated space in both the file and memory.
    - Contrastingly, on line 26 is the `.bss` section. This section stores uninitialized globals which, for my code, are not there. Because of this there is no space allocated for it in the file. However, it does still take up memory

- ELF Segments: *Figure 4*
  - How are sections grouped into segments?
    - Sections, like `.text` and `.data`, define the logical groupings of the program's content. Segments then map these sections into memory at runtime. They are defined, then, in the program headers, not the section headers.
    - For example, each `LOAD` segment combines multiple sections that share similar memory permissions. Looking at Figure 4, Section to Segment mapping:
      - `LOAD 02` is flagged read-only (`R`)
      - `LOAD 03` is flagged as read + execute (`RE`)
      - `LOAD 05` is flagged as read + write (`RW`)
    - Segments group sections by purpose and memory positions to optimize layout for performance and security
      - Executable code, like `.text`, is in a `RE` segment-it is executable but not writable
      - Data sections, like `.data`, are in a `RW` segment-it can be read and modified
      - Read-only data stays in `R` segments
  - What is the purpose of the `PT_LOAD` segment?
    - I have four `PT_LOAD` segments in my output
    - `PT_LOAD` segments are the only ones actually loaded into memory when the binary starts. Each `PT_LOAD` segment will:
      - Map a portion of the file into memory
      - Set memory permissions (`R`, `W`, `X`)
      - Ensure code and data are laid out correctly in RAM
- Symbol Table, `GOT`, and `PLT`:
  - What symbols are present in the binary? *Figure 5*
    - My binary has two main symbol tables: `.dynsym` and `.symtab`
    - `.dynsym` (dynamic symbol table):
      - Used at runtime
      - Contains externally visible symbols, mostly dynamically linked functions
        - `strcpy`, `puts`, `__stack_chk_fail`, `printf`, `__libc_start_main`, `strcmp`, etc.
        - Weak symbols
          - `_ITM_deregisterTMCloneTable`, `__gmon_start__`
        - All entries list Ndx as `UND`, which means they're undefined in this binary and resolved at runtime
    - `.symtab` (full symbol table)
      - Mostly used for debugging and compilation
      - All symbols are static/local functions, variables, section names
      - My user defined functions:
        - `main` at `0x11c9`

        - `show_usage` at `0x1289`
        - `check_password` at `0x12a0`
      - Internal/globals: `_start`, `_init`, `__libc_csu_init`, `_edata`, `_end`, `counter`, etc.
    - Summary
      - `.dnsym` shows imported external symbols
      - show everything, including the program-defined symbols in my code
  - Which functions are dynamically linked? *Figure 5*, *Figure 6*, *Figure 7*
    - From the symbol table, `.dynsym`, we see that the functions `strcpy`, `puts`, `printf`, `strcmp`, and `__stack_chk_fail` are dynamically linked functions. These functions are resolved dynamically and linked at runtime from shared libraries.
    - The `GOT` holds pointers to the dynamically linked functions, which initially are set to point to the corresponding entries in `PLT`.
    - The `PLT` directs the first call of each dynamically linked function to the dynamic linker for resolution (further explained in the next question)
  - How does the `PLT` resolve function addresses? *Figure 6* and *Figure 7*
    - `PLT` contains trampoline code to handle lazy resolution of dynamic functions. On the first call, it redirects through a resolver which then patches the `GOT`. After this, the calls use the patched `GOT` entries directly for faster accessibility
- Additional Analysis:
  - What shared libraries does the binary depend on? *Figure 9*
    - `linux-vdso.so.1`
      - A virtual dynamic shared object (`VDSO`) used by Linux to provide system calls
      - It's a mechanism that avoids slowdowns when making system calls
    - `libc.so.6`
      - The GNU C Library (`libc`), which is a standard library in most Linux systems
      - Provides essential functions for C programs, such as `malloc`, `printf`, `puts`, `strcpy`, `strcmp`, and more
      - The binary relies on `libc.so.6` for core functionality, which is common for most C programs
    - `/lib64/ld-linux-x86-64.so.2`
      - Program that loads other libraries int memory when the program runs
      - Handles startup and shutdown

- What insights can be gained from inspecting the raw ELF contents using xxd? *Figure 10*
    - File identification: the first few bytes (`7f45 4c46`) represents the magic number, `0x7f454c46`, which identifies the file as a proper ELF file. This proves that the file is properly formatted
    - ELF header: metadata on the file can be found as well
        - Architecture: the `02` after the magic number indicates the file is `64-bit`
        - Endianness: The next digits, `01`, indicates little-endian format, which is common for `x86-64` systems
    - On the last line, the `4000 3800` defines the section header table offset. This specifies where the section header table starts in the file. The value (`0x3800`) finds the sections of the program (like `.text` and `.data`)
    - The next digits, `0d00 4000`, represents the flags for the file. This specifies if a file is read-only, executable, writable, etc.

Appendix:

Figure 1: My C Code

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// global variables to store sensitive data

#define ACCESS_MESSAGE "Access granted!\n"
#define ERROR_MESSAGE "Incorrect password.\n"
#define GLOBAL_PASSWORD "SecretPass123"

int counter = 0;

// function declarations

void show_usage();
int check_password(const char *password);

int main(int argc, char *argv[]) {

    if (argc != 2) {
        show_usage();
        return 1;
    }

    // local variable for user input password

    char user_password[64];
    strcpy(user_password, argv[1]);

    // check the password and increment counter on successful access

    if (check_password(user_password)) {
        printf(ACCESS_MESSAGE);
        counter++;
        printf("Access count: %d\n", counter);
    } else {
        printf(ERROR_MESSAGE);
    }

    return 0;
}

void show_usage() {
    printf("Usage: ./program <password>\n");
}

int check_password(const char *password) {

    if (strcmp(password, GLOBAL_PASSWORD) == 0) {
        return 1; // password matches
    } else {
        return 0; // password does not match
    }
}
```

Figure 2: Output when inspect ELF Header

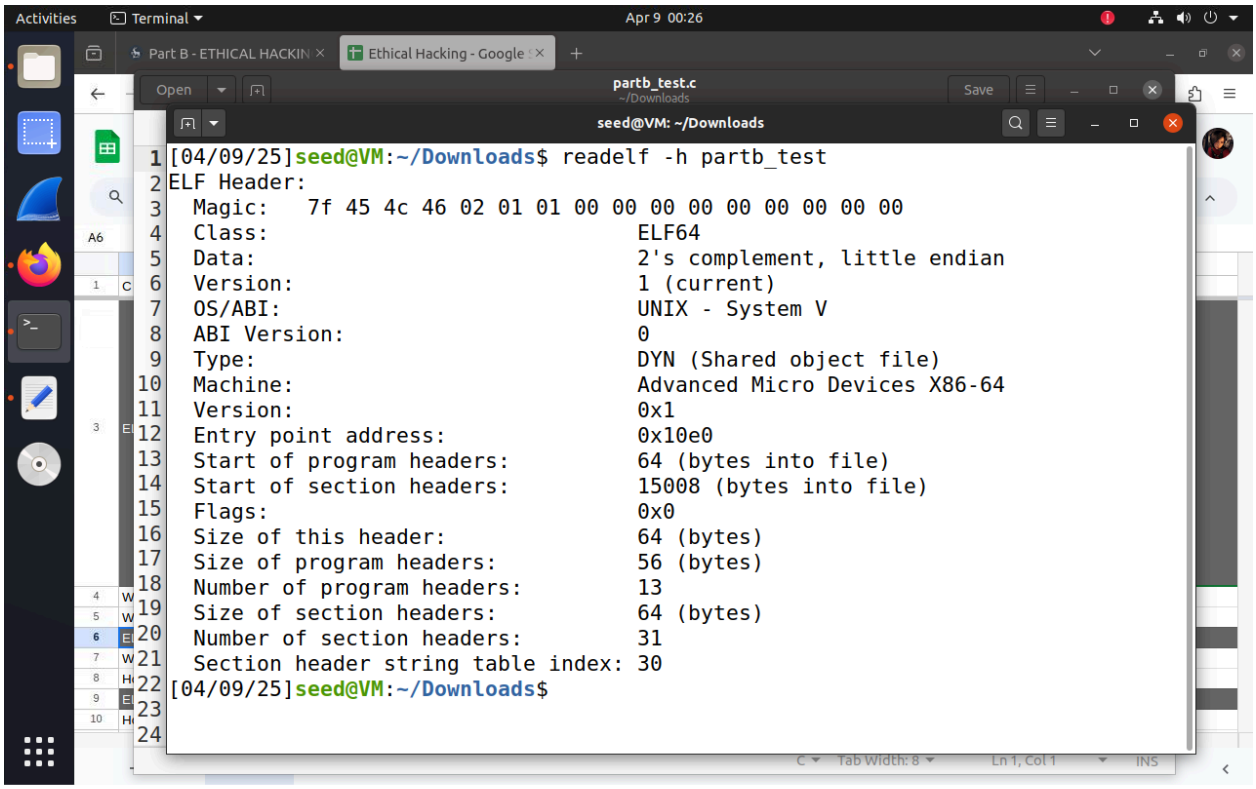
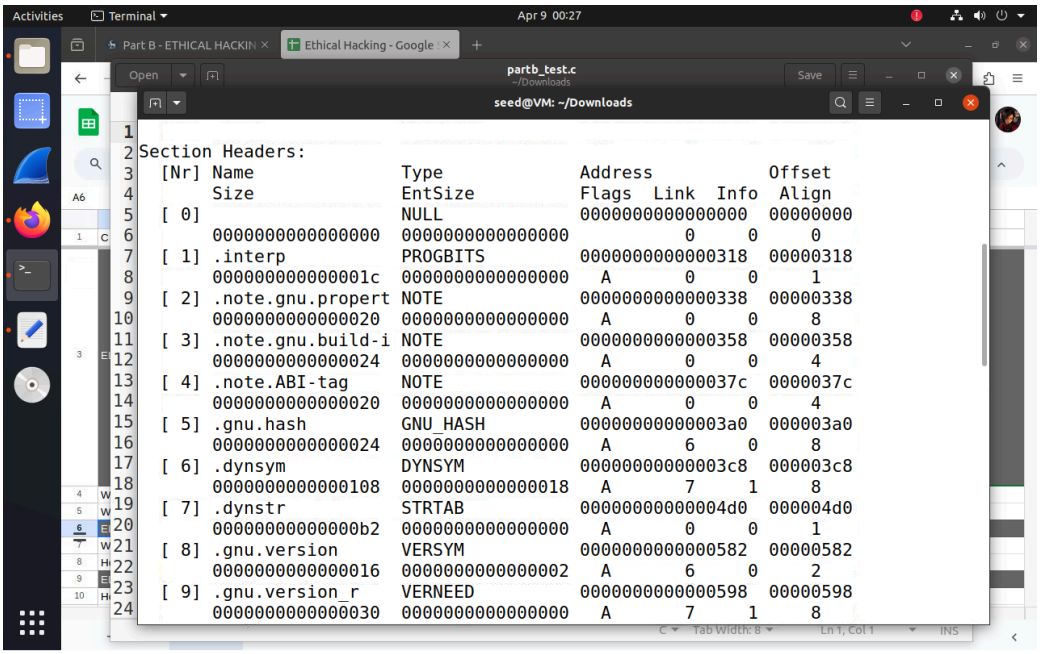
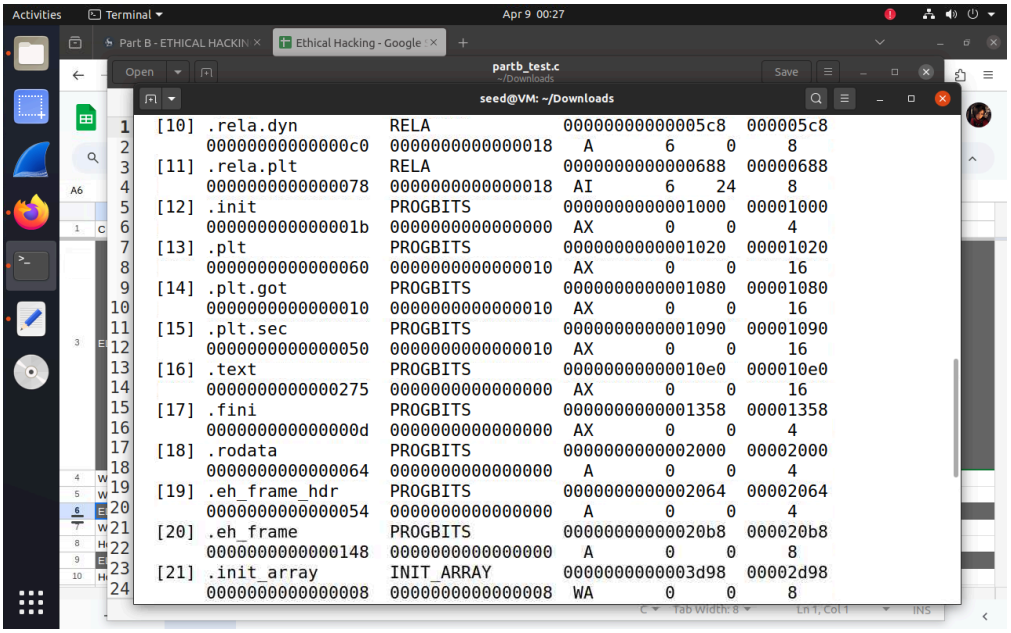


Figure 3: Output when inspecting ELF Sections




Activities
Terminal
Apr 9 00:27
Part B - ETHICAL HACKIN
Ethical Hacking - Google
Open
partb_test.c
~/Downloads
Save
seed@VM: ~/Downloads
[10] .rela.dyn RELA 00000000000005c8 000005c8
00000000000000c0 0000000000000018 A 6 0 8
[11] .rela.plt RELA 0000000000000688 00000688
0000000000000078 0000000000000018 AI 6 24 8
[12] .init PROGBITS 0000000000001000 00001000
000000000000001b 0000000000000000 AX 0 0 4
[13] .plt PROGBITS 0000000000001020 00001020
0000000000000060 0000000000000010 AX 0 0 16
[14] .plt.got PROGBITS 0000000000001080 00001080
0000000000000010 0000000000000010 AX 0 0 16
[15] .plt.sec PROGBITS 0000000000001090 00001090
0000000000000050 0000000000000010 AX 0 0 16
[16] .text PROGBITS 00000000000010e0 000010e0
0000000000000275 0000000000000000 AX 0 0 16
[17] .fini PROGBITS 0000000000001358 00001358
000000000000000d 0000000000000000 AX 0 0 4
[18] .rodata PROGBITS 0000000000002000 00002000
0000000000000064 0000000000000000 A 0 0 4
[19] .eh_frame_hdr PROGBITS 0000000000002064 00002064
0000000000000054 0000000000000000 A 0 0 4
[20] .eh_frame PROGBITS 00000000000020b8 000020b8
0000000000000148 0000000000000000 A 0 0 8
[21] .init_array INIT_ARRAY 0000000000003d98 00002d98
0000000000000008 0000000000000008 WA 0 0 8
Tab Width: 8
Ln 1, Col 1
INS

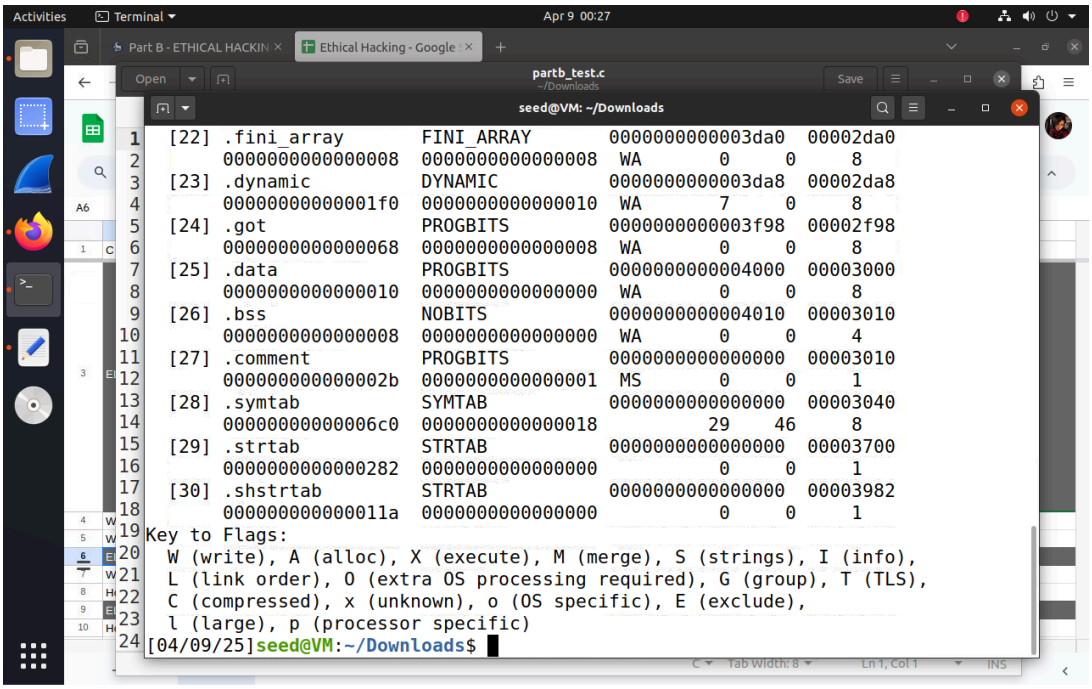
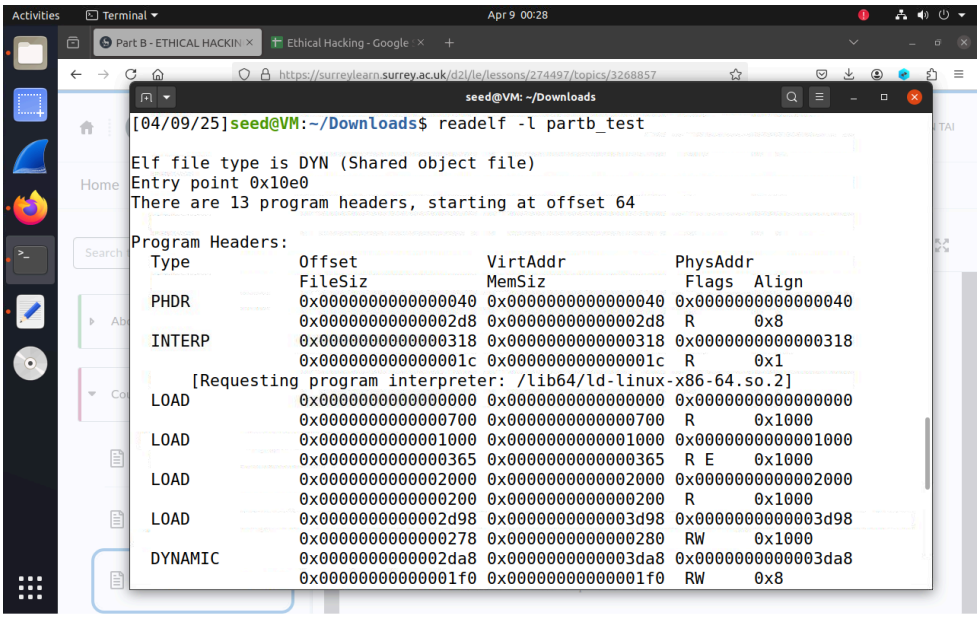
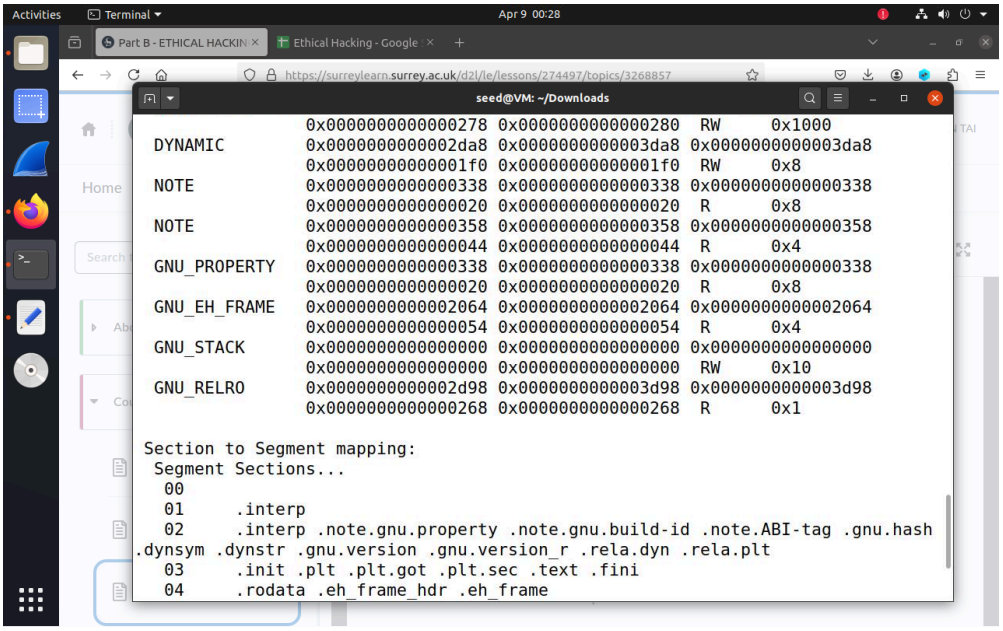

Activities
Terminal
Apr 9 00:27
Part B - ETHICAL HACKIN
Ethical Hacking - Google
Open
partb_test.c
~/Downloads
Save
seed@VM: ~/Downloads
[22] .fini_array FINI_ARRAY 0000000000003da0 00002da0
0000000000000008 0000000000000008 WA 0 0 8
[23] .dynamic DYNAMIC 0000000000003da8 00002da8
00000000000001f0 0000000000000010 WA 7 0 8
[24] .got PROGBITS 0000000000003f98 00002f98
0000000000000068 0000000000000008 WA 0 0 8
[25] .data PROGBITS 0000000000004000 00003000
0000000000000010 0000000000000000 WA 0 0 8
[26] .bss NOBITS 0000000000004010 00003010
0000000000000008 0000000000000000 WA 0 0 4
[27] .comment PROGBITS 0000000000000000 00003010
000000000000002b 0000000000000001 MS 0 0 1
[28] .symtab SYMTAB 0000000000000000 00003040
00000000000006c0 0000000000000018 29 46 8
[29] .strtab STRTAB 0000000000000000 00003700
0000000000000282 0000000000000000 0 0 1
[30] .shstrtab STRTAB 0000000000000000 00003982
000000000000011a 0000000000000000 0 0 1
Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
l (large), p (processor specific)
[04/09/25]seed@VM:~/Downloads$
Tab Width: 8
Ln 1, Col 1
INS

Figure 4: Output when inspecting ELF Segments





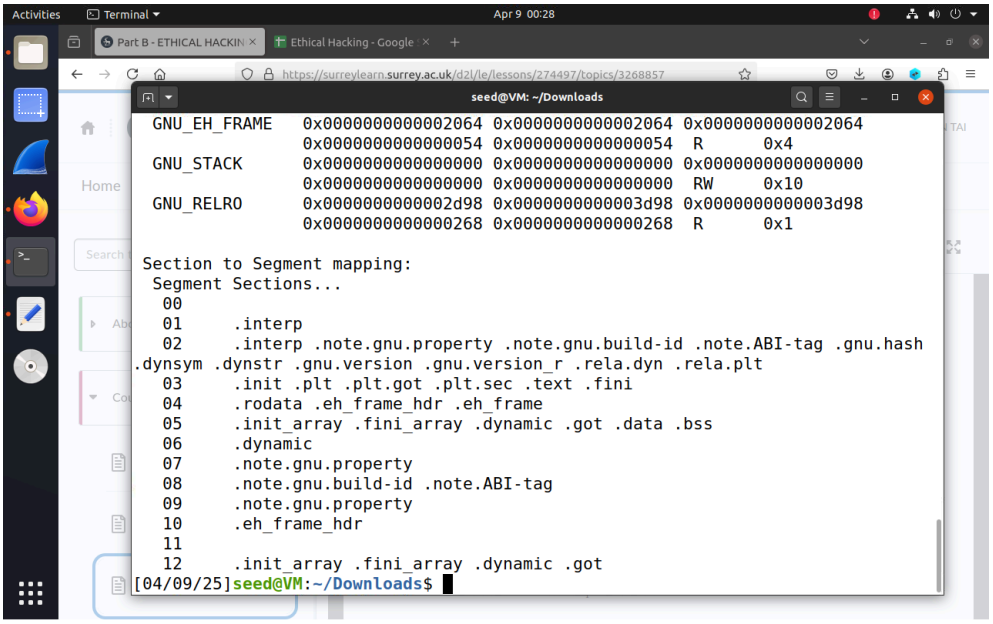
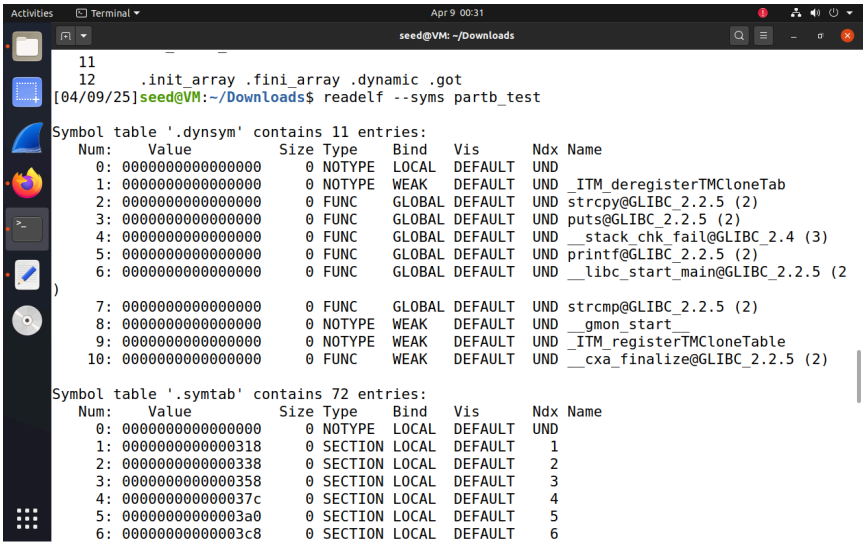
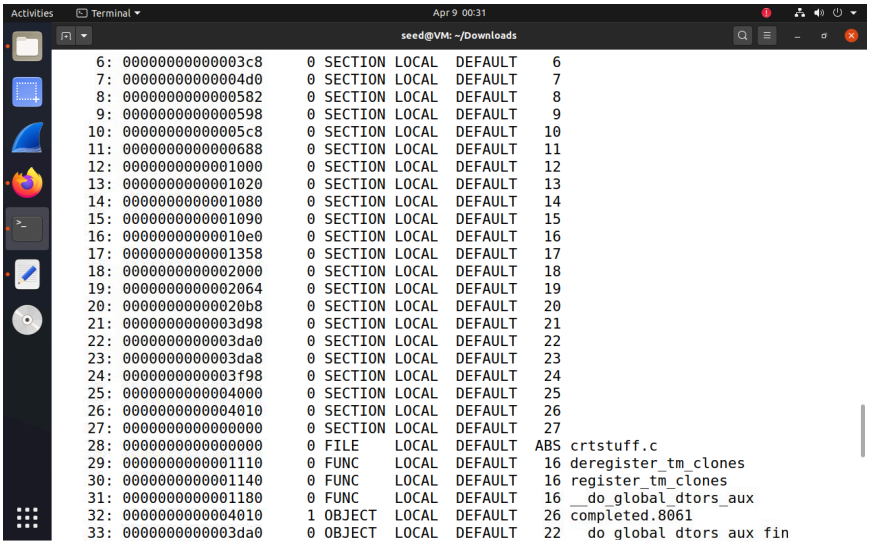
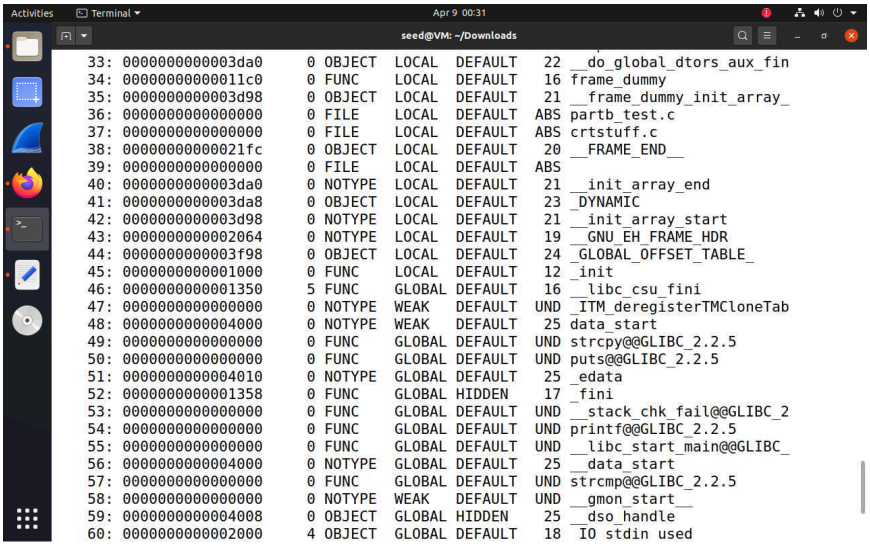


Figure 5: Output when inspecting Symbol Table




Activities
Terminal
Apr 9 00:31
seed@VM: ~/Downloads
6: 00000000000003c8 0 SECTION LOCAL DEFAULT 6
7: 00000000000004d0 0 SECTION LOCAL DEFAULT 7
8: 0000000000000582 0 SECTION LOCAL DEFAULT 8
9: 0000000000000598 0 SECTION LOCAL DEFAULT 9
10: 00000000000005c8 0 SECTION LOCAL DEFAULT 10
11: 0000000000000688 0 SECTION LOCAL DEFAULT 11
12: 0000000000001000 0 SECTION LOCAL DEFAULT 12
13: 0000000000001020 0 SECTION LOCAL DEFAULT 13
14: 0000000000001080 0 SECTION LOCAL DEFAULT 14
15: 0000000000001090 0 SECTION LOCAL DEFAULT 15
16: 00000000000010e0 0 SECTION LOCAL DEFAULT 16
17: 0000000000001358 0 SECTION LOCAL DEFAULT 17
18: 0000000000002000 0 SECTION LOCAL DEFAULT 18
19: 0000000000002064 0 SECTION LOCAL DEFAULT 19
20: 00000000000020b8 0 SECTION LOCAL DEFAULT 20
21: 0000000000003d98 0 SECTION LOCAL DEFAULT 21
22: 0000000000003da0 0 SECTION LOCAL DEFAULT 22
23: 0000000000003da8 0 SECTION LOCAL DEFAULT 23
24: 0000000000003f98 0 SECTION LOCAL DEFAULT 24
25: 0000000000004000 0 SECTION LOCAL DEFAULT 25
26: 0000000000004010 0 SECTION LOCAL DEFAULT 26
27: 0000000000000000 0 SECTION LOCAL DEFAULT 27
28: 0000000000000000 0 FILE LOCAL DEFAULT ABS crtstuff.c
29: 0000000000001110 0 FUNC LOCAL DEFAULT 16 deregister_tm_clones
30: 0000000000001140 0 FUNC LOCAL DEFAULT 16 register_tm_clones
31: 0000000000001180 0 FUNC LOCAL DEFAULT 16 __do_global_dtors_aux
32: 0000000000004010 1 OBJECT LOCAL DEFAULT 26 completed.8061
33: 0000000000003da0 0 OBJECT LOCAL DEFAULT 22 __do_global_dtors_aux_fin


Activities
Terminal
Apr 9 00:31
seed@VM: ~/Downloads
33: 0000000000003da0 0 OBJECT LOCAL DEFAULT 22 __do_global_dtors_aux_fin
34: 00000000000011c0 0 FUNC LOCAL DEFAULT 16 frame_dummy
35: 0000000000003d98 0 OBJECT LOCAL DEFAULT 21 __frame_dummy_init_array_
36: 0000000000000000 0 FILE LOCAL DEFAULT ABS partb_test.c
37: 0000000000000000 0 FILE LOCAL DEFAULT ABS crtstuff.c
38: 00000000000021fc 0 OBJECT LOCAL DEFAULT 20 __FRAME_END__
39: 0000000000000000 0 FILE LOCAL DEFAULT ABS
40: 0000000000003da0 0 NOTYPE LOCAL DEFAULT 21 __init_array_end
41: 0000000000003da8 0 OBJECT LOCAL DEFAULT 23 _DYNAMIC
42: 0000000000003d98 0 NOTYPE LOCAL DEFAULT 21 __init_array_start
43: 0000000000002064 0 NOTYPE LOCAL DEFAULT 19 __GNU_EH_FRAME_HDR
44: 0000000000003f98 0 OBJECT LOCAL DEFAULT 24 _GLOBAL_OFFSET_TABLE_
45: 0000000000001000 0 FUNC LOCAL DEFAULT 12 _init
46: 0000000000001350 5 FUNC GLOBAL DEFAULT 16 __libc_csu_fini
47: 0000000000000000 0 NOTYPE WEAK DEFAULT UND _ITM_deregisterTMCloneTab
48: 0000000000004000 0 NOTYPE WEAK DEFAULT 25 data_start
49: 0000000000000000 0 FUNC GLOBAL DEFAULT UND strcpy@@GLIBC_2.2.5
50: 0000000000000000 0 FUNC GLOBAL DEFAULT UND puts@@GLIBC_2.2.5
51: 0000000000004010 0 NOTYPE GLOBAL DEFAULT 25 _edata
52: 0000000000001358 0 FUNC GLOBAL HIDDEN 17 _fini
53: 0000000000000000 0 FUNC GLOBAL DEFAULT UND __stack_chk_fail@@GLIBC_2
54: 0000000000000000 0 FUNC GLOBAL DEFAULT UND printf@@GLIBC_2.2.5
55: 0000000000000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@@GLIBC_
56: 0000000000004000 0 NOTYPE GLOBAL DEFAULT 25 __data_start
57: 0000000000000000 0 FUNC GLOBAL DEFAULT UND strcmp@@GLIBC_2.2.5
58: 0000000000000000 0 NOTYPE WEAK DEFAULT UND __gmon_start__
59: 0000000000004008 0 OBJECT GLOBAL HIDDEN 25 __dso_handle
60: 0000000000002000 4 OBJECT GLOBAL DEFAULT 18 _IO_stdin_used

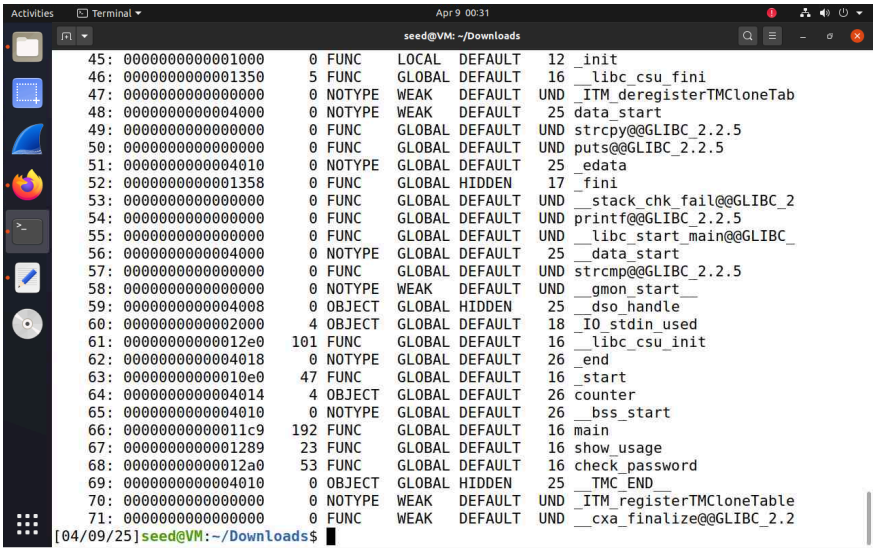

Activities
Terminal
Apr 9 00:31
seed@VM: ~/Downloads
45: 0000000000001000 0 FUNC LOCAL DEFAULT 12 _init
46: 0000000000001350 5 FUNC GLOBAL DEFAULT 16 __libc_csu_fini
47: 0000000000000000 0 NOTYPE WEAK DEFAULT UND _ITM_deregisterTMCloneTab
48: 0000000000004000 0 NOTYPE WEAK DEFAULT 25 data_start
49: 0000000000000000 0 FUNC GLOBAL DEFAULT UND strcpy@@GLIBC_2.2.5
50: 0000000000000000 0 FUNC GLOBAL DEFAULT UND puts@@GLIBC_2.2.5
51: 0000000000004010 0 NOTYPE GLOBAL DEFAULT 25 _edata
52: 0000000000001358 0 FUNC GLOBAL HIDDEN 17 _fini
53: 0000000000000000 0 FUNC GLOBAL DEFAULT UND __stack_chk_fail@@GLIBC_2
54: 0000000000000000 0 FUNC GLOBAL DEFAULT UND printf@@GLIBC_2.2.5
55: 0000000000000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@@GLIBC_
56: 0000000000004000 0 NOTYPE GLOBAL DEFAULT 25 __data_start
57: 0000000000000000 0 FUNC GLOBAL DEFAULT UND strcmp@@GLIBC_2.2.5
58: 0000000000000000 0 NOTYPE WEAK DEFAULT UND __gmon_start__
59: 0000000000004008 0 OBJECT GLOBAL HIDDEN 25 __dso_handle
60: 0000000000002000 4 OBJECT GLOBAL DEFAULT 18 _IO_stdin_used
61: 00000000000012e0 101 FUNC GLOBAL DEFAULT 16 __libc_csu_init
62: 0000000000004018 0 NOTYPE GLOBAL DEFAULT 26 _end
63: 00000000000010e0 47 FUNC GLOBAL DEFAULT 16 _start
64: 0000000000004014 4 OBJECT GLOBAL DEFAULT 26 counter
65: 0000000000004010 0 NOTYPE GLOBAL DEFAULT 26 __bss_start
66: 00000000000011c9 192 FUNC GLOBAL DEFAULT 16 main
67: 0000000000001289 23 FUNC GLOBAL DEFAULT 16 show_usage
68: 00000000000012a0 53 FUNC GLOBAL DEFAULT 16 check_password
69: 0000000000004010 0 OBJECT GLOBAL HIDDEN 25 __TMC_END__
70: 0000000000000000 0 NOTYPE WEAK DEFAULT UND _ITM_registerTMCloneTable
71: 0000000000000000 0 FUNC WEAK DEFAULT UND __cxa_finalize@@GLIBC_2.2
[04/09/25]seed@VM:~/Downloads$

Figure 6: Output when inspecting GOT

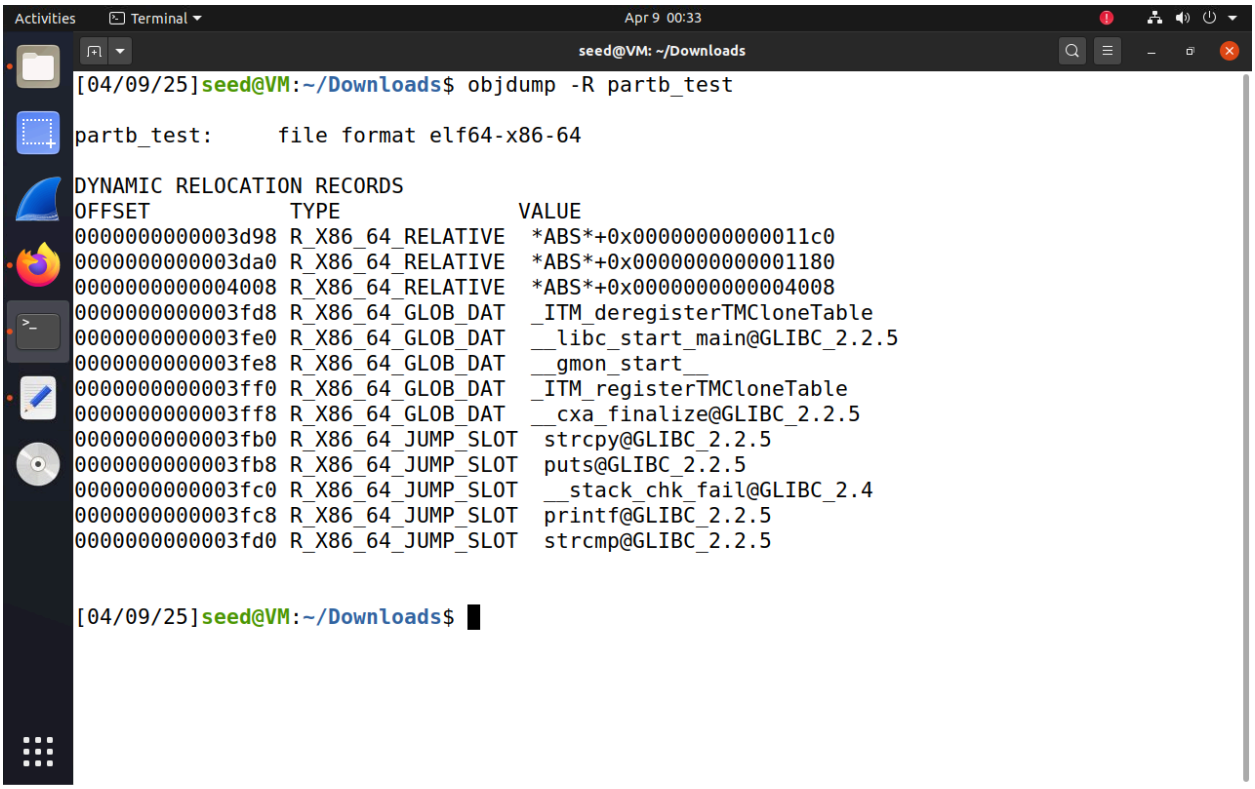


Figure 7: Output when inspecting PLT

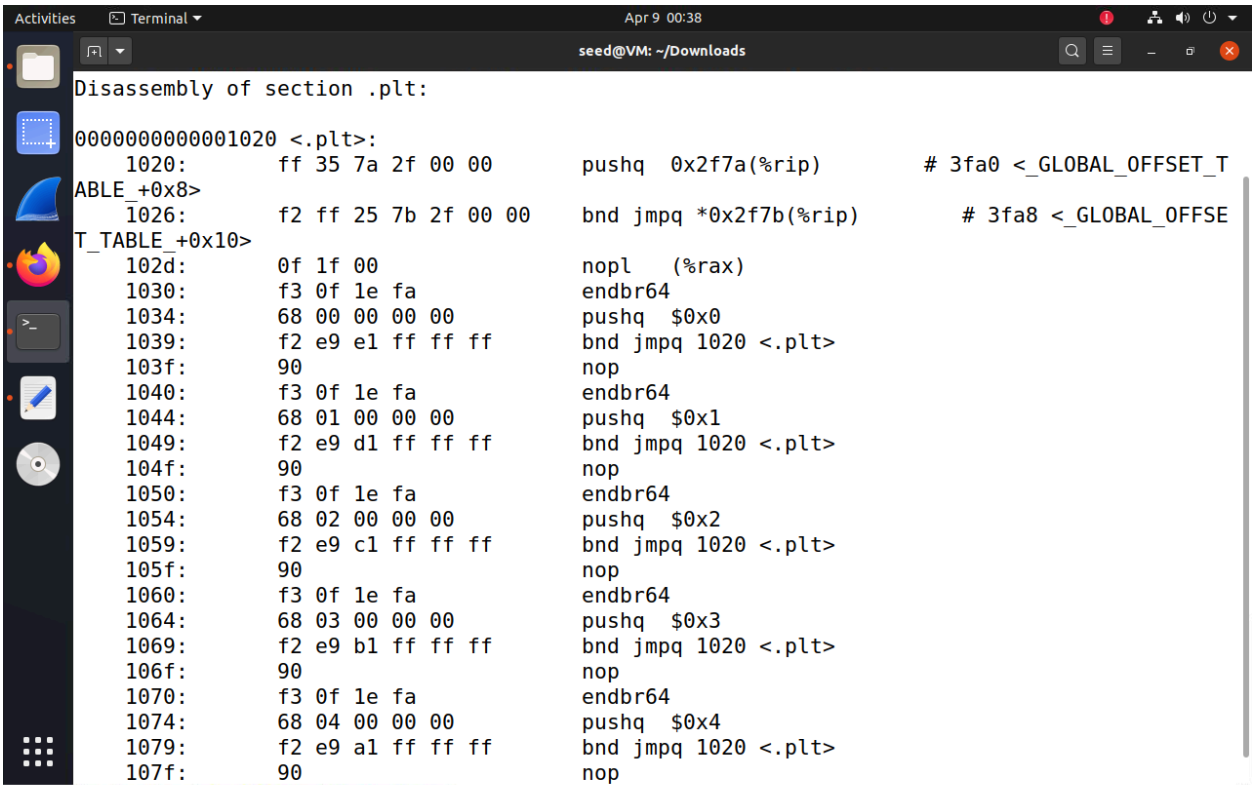
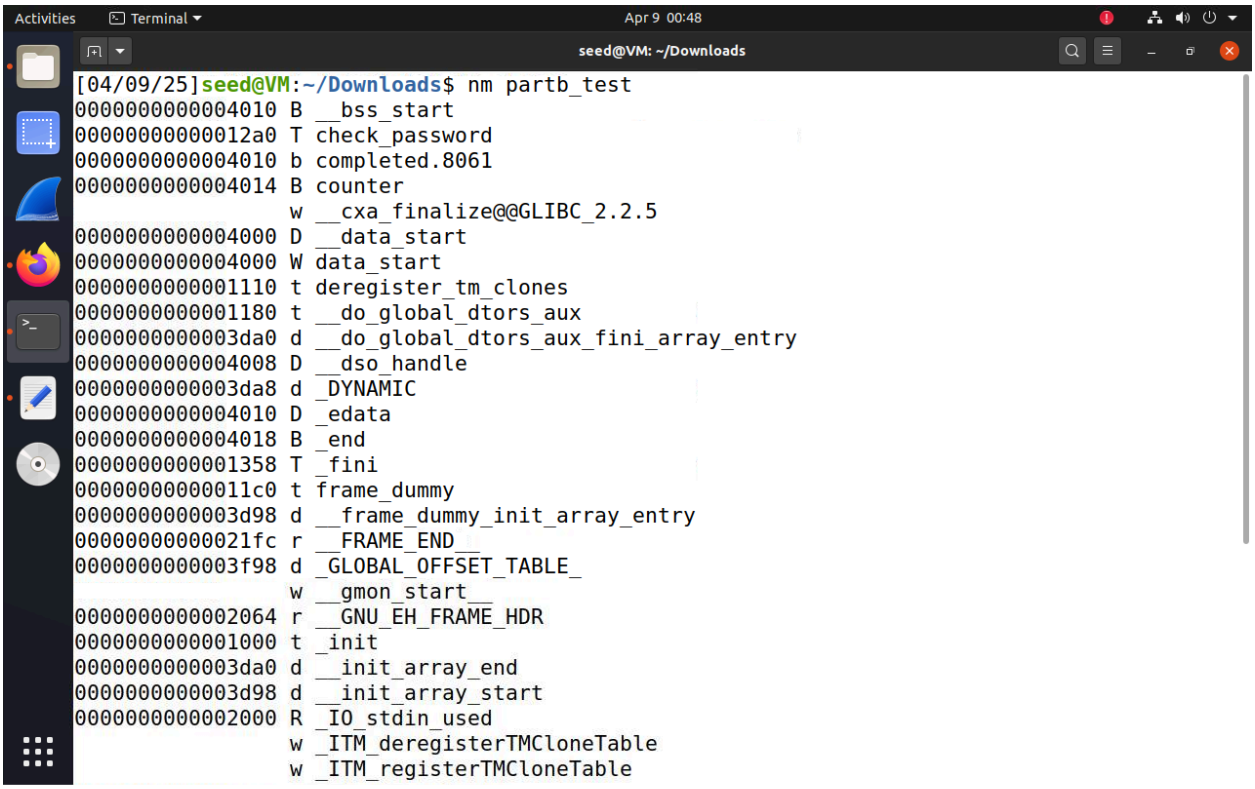


Figure 8: Output when inspecting Symbols



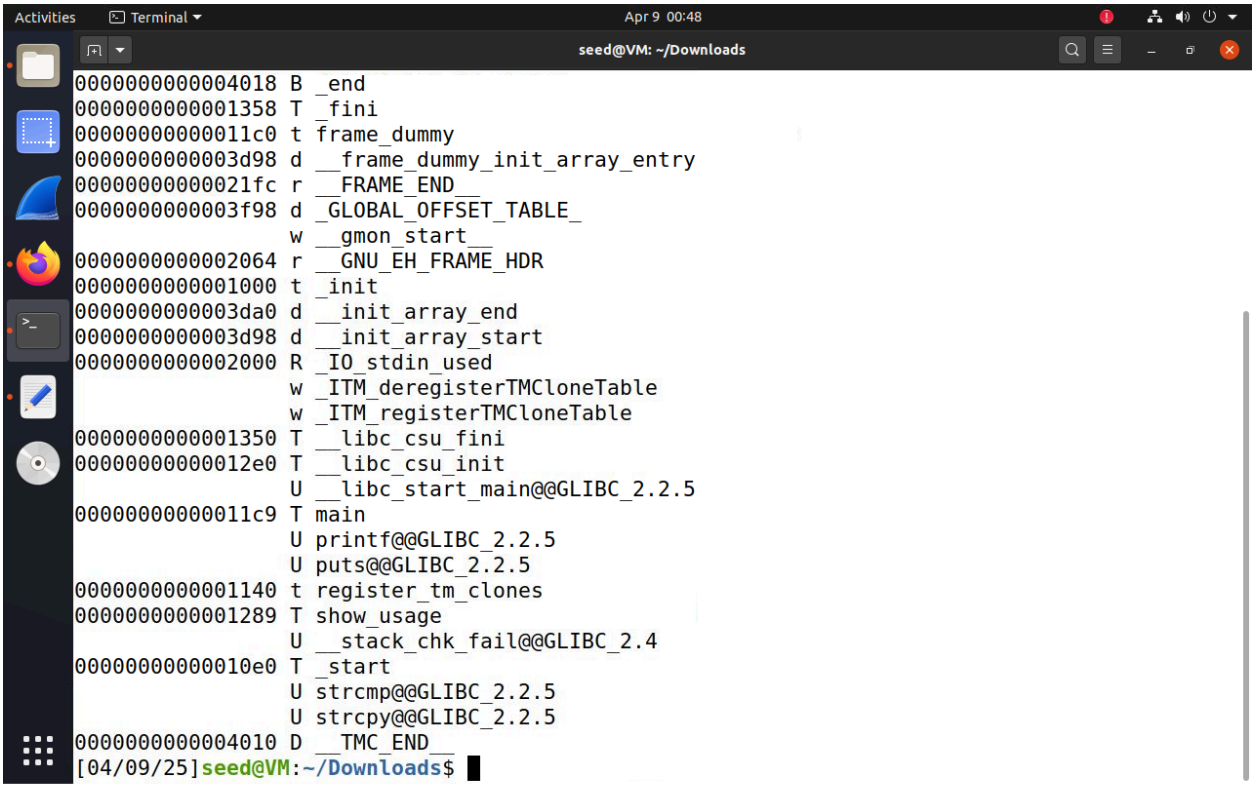


Figure 9: Output when inspecting Shared Libraries

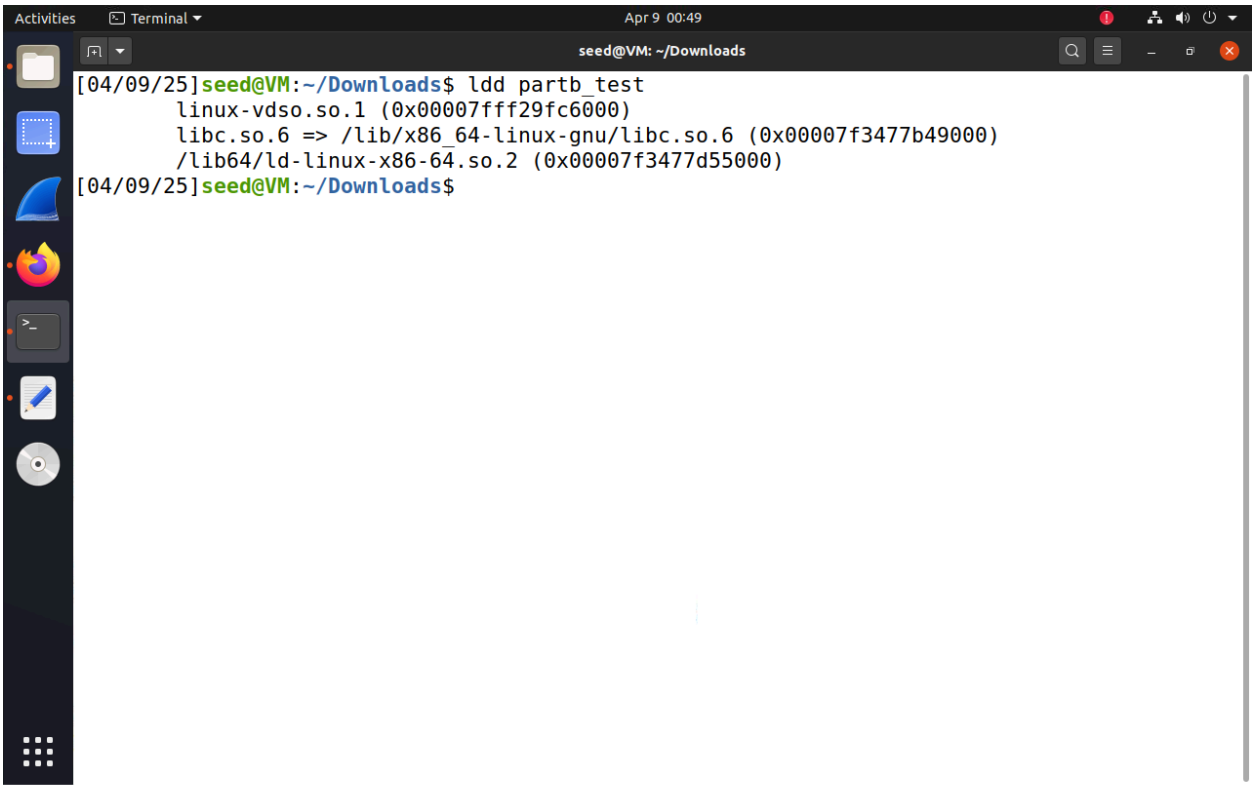


Figure 10: Output when inspecting Binary Contents