

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 `.syms`
 - `.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
 - `.syms` (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
 - `.dynsym` 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 into 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

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 // global variables to store sensitive data
6
7 #define ACCESS_MESSAGE "Access granted!\n"
8 #define ERROR_MESSAGE "Incorrect password.\n"
9 #define GLOBAL_PASSWORD "SecretPass123"
10
11 int counter = 0;
12
13 // function declarations
14
15 void show_usage();
16 int check_password(const char *password);
17
18 int main(int argc, char *argv[]) {
19
20     if (argc != 2) {
21         show_usage();
22         return 1;
23     }
24
25     // local variable for user input password
26
27     char user_password[64];
28     strcpy(user_password, argv[1]);
29
30     // check the password and increment counter on successful access
31
32     if (check_password(user_password)) {
33         printf(ACCESS_MESSAGE);
34         counter++;
35         printf("Access count: %d\n", counter);
36     } else {
37         printf(ERROR_MESSAGE);
38     }
39
40     return 0;
41 }
42
43 void show_usage() {
44     printf("Usage: ./program <password>\n");
45 }
46
47 int check_password(const char *password) {
48
49     if (strcmp(password, GLOBAL_PASSWORD) == 0) {
50         return 1; // password matches
51     } else {
52         return 0; // password does not match
53     }
54 }
```

Figure 2: Output when inspect ELF Header

```

1 [04/09/25]seed@VM:~/Downloads$ readelf -h partb_test
2 ELF Header:
3   Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
4     Class: ELF64
5     Data: 2's complement, little endian
6   Version: 1 (current)
7   OS/ABI: UNIX - System V
8   ABI Version: 0
9   Type: DYN (Shared object file)
10  Machine: Advanced Micro Devices X86-64
11  Version: 0x1
12  Entry point address: 0x10e0
13  Start of program headers: 64 (bytes into file)
14  Start of section headers: 15008 (bytes into file)
15  Flags: 0x0
16  Size of this header: 64 (bytes)
17  Size of program headers: 56 (bytes)
18  Number of program headers: 13
19  Size of section headers: 64 (bytes)
20  Number of section headers: 31
21  Section header string table index: 30
[04/09/25]seed@VM:~/Downloads$

```

Figure 3: Output when inspecting ELF Sections

	[Nr]	Name	Type	Address	Offset	Size	EntSize	Flags	Link	Info	Align
	[0]		NULL	0000000000000000	0000000000000000	00000000			0	0	0
	[1]	.interp	PROGBITS	0000000000000318	0000000000000318	0000000000000318					
	[2]	.note.gnu.property	NOTE	0000000000000038	0000000000000038	0000000000000038					
	[3]	.note.gnu.build-i	NOTE	00000000000000358	00000000000000358	00000000000000358					
	[4]	.note.ABI-tag	NOTE	0000000000000037c	0000000000000037c	0000000000000037c					
	[5]	.gnu.hash	GNU_HASH	00000000000003a0	00000000000003a0	00000000000003a0					
	[6]	.dynsym	DYNSYM	00000000000003c8	00000000000003c8	00000000000003c8					
	[7]	.dynstr	STRTAB	00000000000004d0	00000000000004d0	00000000000004d0					
	[8]	.gnu.version	VERSYM	0000000000000582	0000000000000582	0000000000000582					
	[9]	.gnu.version_r	VERNEED	0000000000000598	0000000000000598	0000000000000598					
				0000000000000000	0000000000000000	0000000000000000					

```
partb_test.c
~/Downloads
seed@VM: ~/Downloads

[10] .rela.dyn      RELA    0000000000000005c8 000005c8
[11] .rela.plt     RELA    0000000000000000688 00000688
[12] .init        PROGBITS 0000000000001000 00001000
[13] .plt         PROGBITS 0000000000001020 00001020
[14] .plt.got     PROGBITS 0000000000001080 00001080
[15] .plt.sec     PROGBITS 0000000000001090 00001090
[16] .text        PROGBITS 00000000000010e0 000010e0
[17] .fini        PROGBITS 0000000000001358 00001358
[18] .rodata      PROGBITS 0000000000002000 00002000
[19] .eh_frame_hdr PROGBITS 0000000000002064 00002064
[20] .eh_frame    PROGBITS 00000000000020b8 000020b8
[21] .init_array  INIT_ARRAY 0000000000003d98 00002d98
[22] .fini_array  FINI_ARRAY 0000000000003da0 00002da0
[23] .dynamic     DYNAMIC  0000000000003da8 00002da8
[24] .got         PROGBITS 0000000000003f98 00002f98
[25] .data        PROGBITS 0000000000004000 00003000
[26] .bss         NOBITS   0000000000004010 00003010
[27] .comment     PROGBITS 0000000000004010 00003010
[28] .symtab     SYMTAB   0000000000004010 00003040
[29] .strtab     STRTAB   0000000000004010 00003700
[30] .shstrtab   STRTAB   0000000000004010 00003982
```

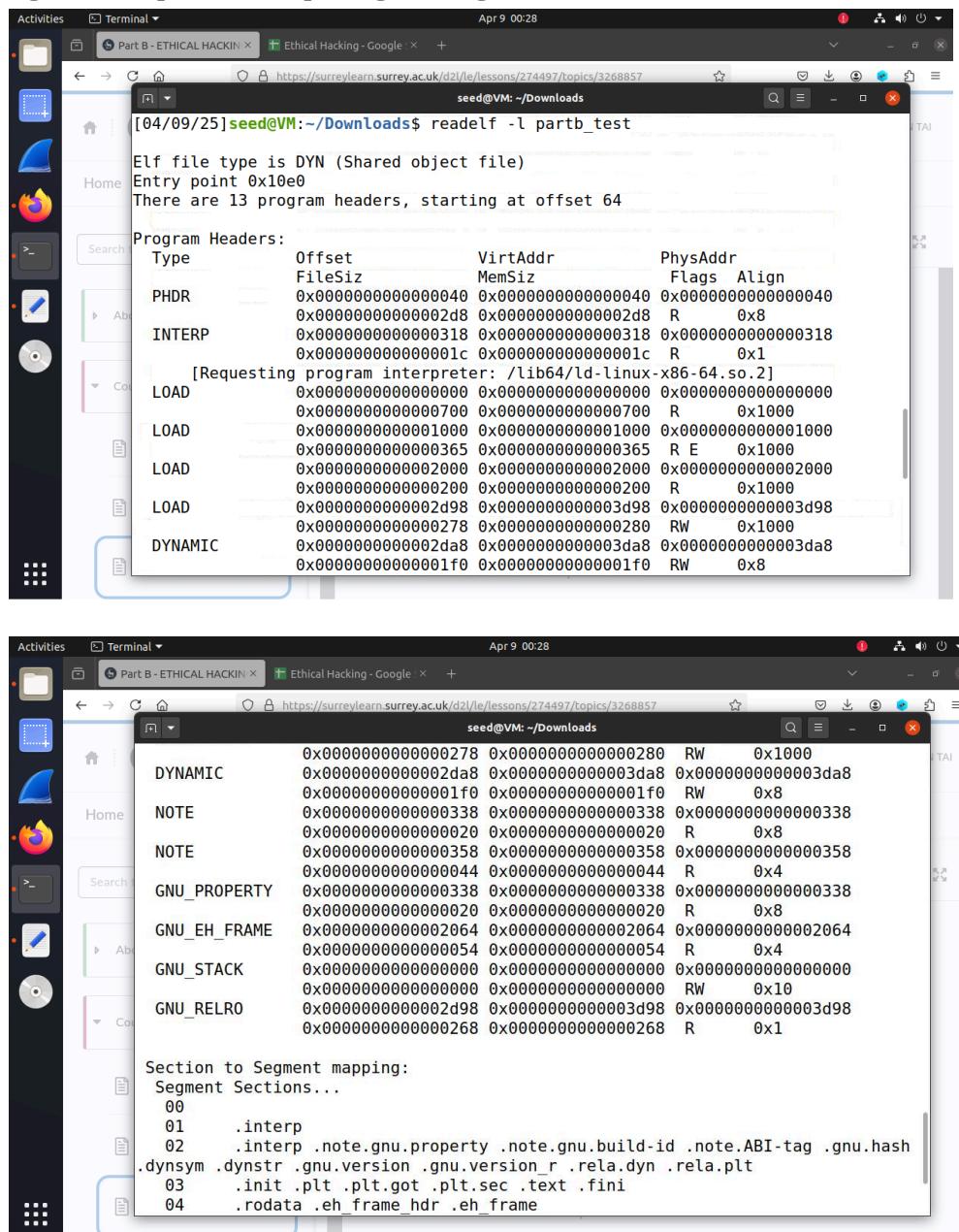
```
partb_test.c
~/Downloads
seed@VM: ~/Downloads

[22] .fini_array  FINI_ARRAY 0000000000003da0 00002da0
[23] .dynamic     DYNAMIC   0000000000003da8 00002da8
[24] .got         PROGBITS  0000000000003f98 00002f98
[25] .data        PROGBITS  0000000000004000 00003000
[26] .bss         NOBITS   0000000000004010 00003010
[27] .comment     PROGBITS  0000000000004010 00003010
[28] .symtab     SYMTAB   0000000000004010 00003040
[29] .strtab     STRTAB   0000000000004010 00003700
[30] .shstrtab   STRTAB   0000000000004010 00003982

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$
```

Figure 4: Output when inspecting ELF Segments



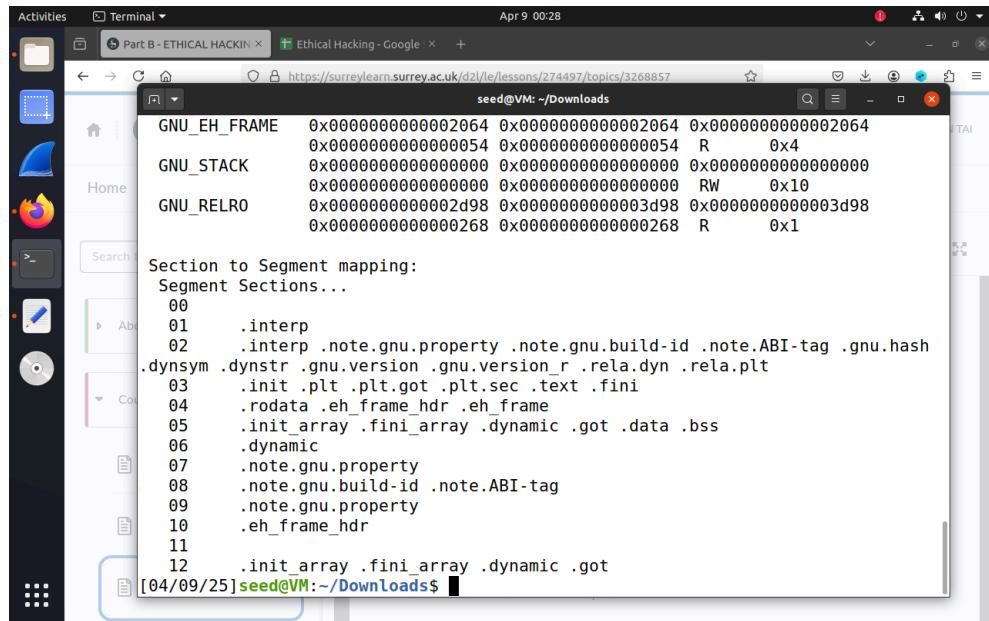
The screenshot shows a terminal window titled 'seed@VM: ~/Downloads' running on a Linux desktop environment. The terminal displays the output of the command 'readelf -l partb_test'. The output includes information about the ELF file type, entry point, program headers, and dynamic sections.

```
[04/09/25]seed@VM:~/Downloads$ readelf -l partb_test
Elf file type is DYN (Shared object file)
Entry point 0x10e0
There are 13 program headers, starting at offset 64

Program Headers:
Type          Offset      VirtAddr      PhysAddr
FileSiz      MemSiz      Flags Align
PHDR          0x0000000000000040 0x0000000000000040 0x0000000000000040
              0x00000000000002d8 0x00000000000002d8 R      0x8
INTERP        0x0000000000000318 0x0000000000000318 0x0000000000000318
              0x000000000000001c 0x000000000000001c R      0x1
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
LOAD          0x0000000000000000 0x0000000000000000 0x0000000000000000
              0x0000000000000700 0x0000000000000700 R      0x1000
LOAD          0x00000000000001000 0x00000000000001000 0x00000000000001000
              0x0000000000000365 0x0000000000000365 R E    0x1000
LOAD          0x00000000000002000 0x00000000000002000 0x00000000000002000
              0x0000000000000200 0x0000000000000200 R      0x1000
LOAD          0x00000000000002d98 0x00000000000003d98 0x00000000000003d98
              0x0000000000000278 0x0000000000000280 RW    0x1000
DYNAMIC       0x00000000000002da8 0x00000000000003da8 0x00000000000003da8
              0x00000000000001f0 0x00000000000001f0 RW    0x8

Dynamic Section:
DYNAMIC       0x00000000000000278 0x00000000000000280 RW    0x1000
              0x00000000000002da8 0x00000000000003da8 0x00000000000003da8
              0x00000000000001f0 0x00000000000001f0 RW    0x8
NOTE          0x00000000000000338 0x00000000000000338 0x00000000000000338
              0x0000000000000020 0x0000000000000020 R      0x8
NOTE          0x00000000000000358 0x00000000000000358 0x00000000000000358
              0x0000000000000044 0x0000000000000044 R      0x4
GNU_PROPERTY   0x00000000000000338 0x00000000000000338 0x00000000000000338
              0x0000000000000020 0x0000000000000020 R      0x8
GNU_EH_FRAME   0x0000000000002064 0x0000000000002064 0x0000000000002064
              0x0000000000000054 0x0000000000000054 R      0x4
GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
              0x0000000000000000 0x0000000000000000 RW    0x10
GNU_RELRO      0x00000000000002d98 0x00000000000003d98 0x00000000000003d98
              0x0000000000000268 0x0000000000000268 R      0x1

Section to Segment mapping:
Segment Sections...
 00
 01  .interp
 02  .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag .gnu.hash
.dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt
 03  .init .plt .plt.got .plt.sec .text .fini
 04  .rodata .eh_frame_hdr .eh_frame
```



```

Activities Terminal Apr 9 00:28
Part B - ETHICAL HACKIN X Ethical Hacking - Google +
https://surreylearn.surrey.ac.uk/dz/le/lessons/274497/topics/3268857
seed@VM: ~/Downloads

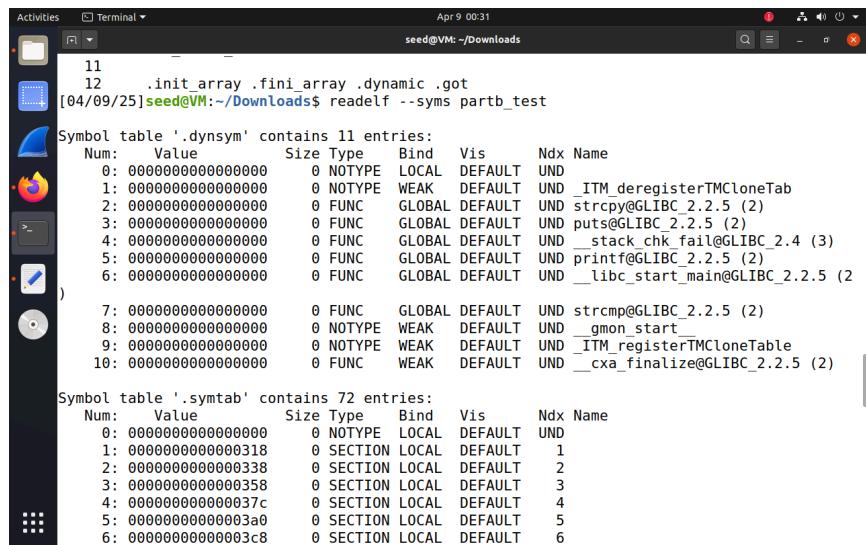
GNU_EH_FRAME 0x0000000000002064 0x0000000000002064 0x0000000000002064
0x0000000000000054 0x0000000000000054 R 0x4
GNU_STACK 0x0000000000000000 0x0000000000000000 0x0000000000000000
0x0000000000000000 0x0000000000000000 RW 0x10
GNU_RELRO 0x000000000002d98 0x0000000000003d98 0x0000000000003d98
0x000000000000268 0x000000000000268 R 0x1

Section to Segment mapping:
Segment Sections...
 00
 01 .interp
 02 .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag .gnu.hash
.dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt
 03 .init .plt .plt.got .plt.sec .text .fini
 04 .rodata .eh_frame_hdr .eh_frame
 05 .init_array .fini_array .dynamic .got .data .bss
 06 .dynamic
 07 .note.gnu.property
 08 .note.gnu.build-id .note.ABI-tag
 09 .note.gnu.property
 10 .eh_frame_hdr
 11
 12 .init_array .fini_array .dynamic .got

[04/09/25]seed@VM:~/Downloads$ 

```

Figure 5: Output when inspecting Symbol Table



```

Activities Terminal Apr 9 00:31
seed@VM: ~/Downloads

11
12 .init_array .fini_array .dynamic .got
[04/09/25]seed@VM:~/Downloads$ readelf --syms partb_test

Symbol table '.dynsym' contains 11 entries:
Num: Value          Size Type Bind Vis Ndx Name
 0: 0000000000000000    0 NOTYPE LOCAL DEFAULT UND 
 1: 0000000000000000    0 NOTYPE WEAK  DEFAULT UND _ITM_deregisterTMCloneTab
 2: 0000000000000000    0 FUNC   GLOBAL DEFAULT UND strcpy@GLIBC_2.2.5 (2)
 3: 0000000000000000    0 FUNC   GLOBAL DEFAULT UND puts@GLIBC_2.2.5 (2)
 4: 0000000000000000    0 FUNC   GLOBAL DEFAULT UND __stack_chk_fail@GLIBC_2.4 (3)
 5: 0000000000000000    0 FUNC   GLOBAL DEFAULT UND printf@GLIBC_2.2.5 (2)
 6: 0000000000000000    0 FUNC   GLOBAL DEFAULT UND __libc_start_main@GLIBC_2.2.5 (2)
 7: 0000000000000000    0 FUNC   GLOBAL DEFAULT UND strcmp@GLIBC_2.2.5 (2)
 8: 0000000000000000    0 NOTYPE WEAK  DEFAULT UND __gmon_start__
 9: 0000000000000000    0 NOTYPE WEAK  DEFAULT UND _ITM_registerTMCloneTable
10: 0000000000000000    0 FUNC   WEAK  DEFAULT UND __cxa_finalize@GLIBC_2.2.5 (2)

Symbol table '.symtab' contains 72 entries:
Num: Value          Size Type Bind Vis Ndx Name
 0: 0000000000000000    0 NOTYPE LOCAL DEFAULT UND 
 1: 0000000000000318    0 SECTION LOCAL DEFAULT 1
 2: 0000000000000338    0 SECTION LOCAL DEFAULT 2
 3: 0000000000000358    0 SECTION LOCAL DEFAULT 3
 4: 000000000000037c    0 SECTION LOCAL DEFAULT 4
 5: 00000000000003a0    0 SECTION LOCAL DEFAULT 5
 6: 00000000000003c8    0 SECTION LOCAL DEFAULT 6

```

```
Activities Terminal ▾ Apr 9 00:31
seed@VM: ~/Downloads
6: 000000000000003c8 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: 00000000000001000 0 SECTION LOCAL DEFAULT 12
13: 00000000000001020 0 SECTION LOCAL DEFAULT 13
14: 00000000000001080 0 SECTION LOCAL DEFAULT 14
15: 00000000000001090 0 SECTION LOCAL DEFAULT 15
16: 000000000000010e0 0 SECTION LOCAL DEFAULT 16
17: 00000000000001358 0 SECTION LOCAL DEFAULT 17
18: 00000000000002000 0 SECTION LOCAL DEFAULT 18
19: 00000000000002064 0 SECTION LOCAL DEFAULT 19
20: 000000000000020b8 0 SECTION LOCAL DEFAULT 20
21: 00000000000003d98 0 SECTION LOCAL DEFAULT 21
22: 00000000000003da0 0 SECTION LOCAL DEFAULT 22
23: 00000000000003da8 0 SECTION LOCAL DEFAULT 23
24: 00000000000003f98 0 SECTION LOCAL DEFAULT 24
25: 00000000000004000 0 SECTION LOCAL DEFAULT 25
26: 00000000000004010 0 SECTION LOCAL DEFAULT 26
27: 00000000000000000 0 SECTION LOCAL DEFAULT 27
28: 00000000000000000 0 FILE LOCAL DEFAULT ABS crtstuff.c
29: 0000000000000110 0 FUNC LOCAL DEFAULT 16 deregister_tm_clones
30: 00000000000001140 0 FUNC LOCAL DEFAULT 16 register_tm_clones
31: 00000000000001180 0 FUNC LOCAL DEFAULT 16 __do_global_dtors_aux
32: 00000000000004010 1 OBJECT LOCAL DEFAULT 26 completed.8061
33: 00000000000003da0 0 OBJECT LOCAL DEFAULT 22 do_global_dtors_aux fin
```

```
Activities Terminal ▾ Apr 9 00:31
seed@VM: ~/Downloads
33: 00000000000003da0 0 OBJECT LOCAL DEFAULT 22 __do_global_dtors_aux_fin
34: 000000000000011c0 0 FUNC LOCAL DEFAULT 16 __frame_dummy
35: 00000000000003d98 0 OBJECT LOCAL DEFAULT 21 __frame_dummy_init_array_
36: 00000000000000000 0 FILE LOCAL DEFAULT ABS partb_test.c
37: 00000000000000000 0 FILE LOCAL DEFAULT ABS crtstuff.c
38: 000000000000021fc 0 OBJECT LOCAL DEFAULT 20 __FRAME_END__
39: 00000000000000000 0 FILE LOCAL DEFAULT ABS
40: 00000000000003da0 0 NOTYPE LOCAL DEFAULT 21 __init_array_end
41: 00000000000003d8 0 OBJECT LOCAL DEFAULT 23 DYNAMIC
42: 00000000000003d98 0 NOTYPE LOCAL DEFAULT 21 __init_array_start
43: 00000000000002064 0 NOTYPE LOCAL DEFAULT 19 __GNU_EH_FRAME_HDR
44: 00000000000003f98 0 OBJECT LOCAL DEFAULT 24 __GLOBAL_OFFSET_TABLE__
45: 00000000000001000 0 FUNC LOCAL DEFAULT 12 __init
46: 00000000000001350 5 FUNC GLOBAL DEFAULT 16 __libc_csu_fini
47: 00000000000000000 0 NOTYPE WEAK DEFAULT UND __ITM_deregisterTMCloneTab
48: 00000000000004000 0 NOTYPE WEAK DEFAULT 25 __data_start
49: 00000000000000000 0 FUNC GLOBAL DEFAULT UND strcpy@GLIBC_2.2.5
50: 00000000000000000 0 FUNC GLOBAL DEFAULT UND puts@GLIBC_2.2.5
51: 00000000000004010 0 NOTYPE GLOBAL DEFAULT 25 __edata
52: 00000000000001358 0 FUNC GLOBAL HIDDEN 17 __fini
53: 00000000000000000 0 FUNC GLOBAL DEFAULT UND __stack_chk_fail@GLIBC_2
54: 00000000000000000 0 FUNC GLOBAL DEFAULT UND printf@GLIBC_2.2.5
55: 00000000000000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@GLIBC_
56: 00000000000004000 0 NOTYPE GLOBAL DEFAULT 25 __data_start
57: 00000000000000000 0 FUNC GLOBAL DEFAULT UND strcmp@GLIBC_2.2.5
58: 00000000000000000 0 NOTYPE WEAK DEFAULT UND __gmon_start__
59: 00000000000004008 0 OBJECT GLOBAL HIDDEN 25 __dso_handle
60: 00000000000002000 4 OBJECT GLOBAL DEFAULT 18 __I0_stdin_used
61: 000000000000012e0 101 FUNC GLOBAL DEFAULT 16 __libc_csu_init
62: 00000000000004018 0 NOTYPE GLOBAL DEFAULT 26 __end
63: 000000000000010e0 47 FUNC GLOBAL DEFAULT 16 __start
64: 00000000000004014 4 OBJECT GLOBAL DEFAULT 26 counter
65: 00000000000004010 0 NOTYPE GLOBAL DEFAULT 26 __bss_start
66: 000000000000011c9 192 FUNC GLOBAL DEFAULT 16 main
67: 00000000000001289 23 FUNC GLOBAL DEFAULT 16 show_usage
68: 000000000000012a0 53 FUNC GLOBAL DEFAULT 16 check_password
69: 00000000000004010 0 OBJECT GLOBAL HIDDEN 25 __TMC_END__
70: 00000000000000000 0 NOTYPE WEAK DEFAULT UND __ITM_registerTMCloneTable
71: 00000000000000000 0 FUNC WEAK DEFAULT UND __cxa_finalize@GLIBC_2.2
```

```
Activities Terminal ▾ Apr 9 00:31
seed@VM: ~/Downloads
45: 00000000000001000 0 FUNC LOCAL DEFAULT 12 __init
46: 00000000000001350 5 FUNC GLOBAL DEFAULT 16 __libc_csu_fini
47: 00000000000000000 0 NOTYPE WEAK DEFAULT UND __ITM_deregisterTMCloneTab
48: 00000000000004000 0 NOTYPE WEAK DEFAULT 25 __data_start
49: 00000000000000000 0 FUNC GLOBAL DEFAULT UND strcpy@GLIBC_2.2.5
50: 00000000000000000 0 FUNC GLOBAL DEFAULT UND puts@GLIBC_2.2.5
51: 00000000000004010 0 NOTYPE GLOBAL DEFAULT 25 __edata
52: 00000000000001358 0 FUNC GLOBAL HIDDEN 17 __fini
53: 00000000000000000 0 FUNC GLOBAL DEFAULT UND __stack_chk_fail@GLIBC_2
54: 00000000000000000 0 FUNC GLOBAL DEFAULT UND printf@GLIBC_2.2.5
55: 00000000000000000 0 FUNC GLOBAL DEFAULT UND __libc_start_main@GLIBC_
56: 00000000000004000 0 NOTYPE GLOBAL DEFAULT 25 __data_start
57: 00000000000000000 0 FUNC GLOBAL DEFAULT UND strcmp@GLIBC_2.2.5
58: 00000000000000000 0 NOTYPE WEAK DEFAULT UND __gmon_start__
59: 00000000000004008 0 OBJECT GLOBAL HIDDEN 25 __dso_handle
60: 00000000000002000 4 OBJECT GLOBAL DEFAULT 18 __I0_stdin_used
61: 000000000000012e0 101 FUNC GLOBAL DEFAULT 16 __libc_csu_init
62: 00000000000004018 0 NOTYPE GLOBAL DEFAULT 26 __end
63: 000000000000010e0 47 FUNC GLOBAL DEFAULT 16 __start
64: 00000000000004014 4 OBJECT GLOBAL DEFAULT 26 counter
65: 00000000000004010 0 NOTYPE GLOBAL DEFAULT 26 __bss_start
66: 000000000000011c9 192 FUNC GLOBAL DEFAULT 16 main
67: 00000000000001289 23 FUNC GLOBAL DEFAULT 16 show_usage
68: 000000000000012a0 53 FUNC GLOBAL DEFAULT 16 check_password
69: 00000000000004010 0 OBJECT GLOBAL HIDDEN 25 __TMC_END__
70: 00000000000000000 0 NOTYPE WEAK DEFAULT UND __ITM_registerTMCloneTable
71: 00000000000000000 0 FUNC WEAK DEFAULT UND __cxa_finalize@GLIBC_2.2
```

[04/09/25] seed@VM:~/Downloads\$

Figure 6: Output when inspecting GOT

```

[04/09/25] seed@VM:~/Downloads$ objdump -R partb_test

partb_test:      file format elf64-x86-64

DYNAMIC RELOCATION RECORDS
OFFSET           TYPE              VALUE
0000000000003d98 R_X86_64_RELATIVE  *ABS*+0x000000000000011c0
0000000000003da0 R_X86_64_RELATIVE  *ABS*+0x00000000000001180
0000000000004008 R_X86_64_RELATIVE  *ABS*+0x00000000000004008
0000000000003fd8 R_X86_64_GLOB_DAT _ITM_deregisterTMCloneTable
0000000000003fe0 R_X86_64_GLOB_DAT __libc_start_main@GLIBC_2.2.5
0000000000003fe8 R_X86_64_GLOB_DAT __gmon_start__
0000000000003ff0 R_X86_64_GLOB_DAT __ITM_registerTMCloneTable
0000000000003ff8 R_X86_64_GLOB_DAT __cxa_finalize@GLIBC_2.2.5
0000000000003fb0 R_X86_64_JUMP_SLOT strcpy@GLIBC_2.2.5
0000000000003fb8 R_X86_64_JUMP_SLOT puts@GLIBC_2.2.5
0000000000003fc0 R_X86_64_JUMP_SLOT __stack_chk_fail@GLIBC_2.4
0000000000003fc8 R_X86_64_JUMP_SLOT printf@GLIBC_2.2.5
0000000000003fd0 R_X86_64_JUMP_SLOT strcmp@GLIBC_2.2.5

[04/09/25] seed@VM:~/Downloads$ 

```

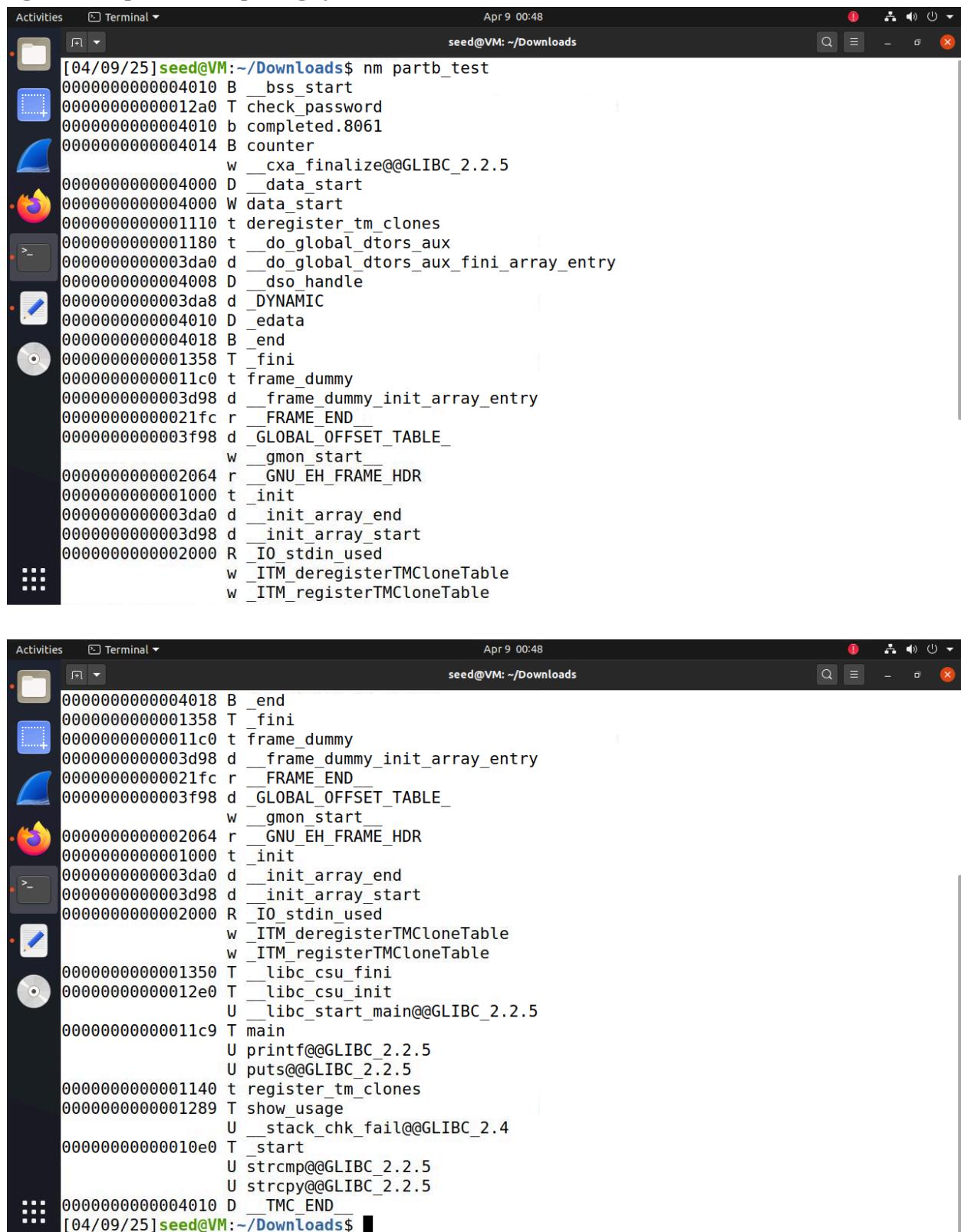
Figure 7: Output when inspecting PLT

```

Disassembly of section .plt:
0000000000001020 <.plt>:
 1020: ff 35 7a 2f 00 00      pushq  0x2f7a(%rip)    # 3fa0 <_GLOBAL_OFFSET_T
ABLE_+0x8>
 1026: f2 ff 25 7b 2f 00 00  bnd jmpq *0x2f7b(%rip)    # 3fa8 <_GLOBAL_OFFSET_T
T_TABLE_+0x10>
 102d: 0f 1f 00                nopl   (%rax)
 1030: f3 0f 1e fa             endbr64
 1034: 68 00 00 00 00          pushq  $0x0
 1039: f2 e9 e1 ff ff ff     bnd jmpq 1020 <.plt>
 103f: 90                      nop
 1040: f3 0f 1e fa             endbr64
 1044: 68 01 00 00 00          pushq  $0x1
 1049: f2 e9 d1 ff ff ff     bnd jmpq 1020 <.plt>
 104f: 90                      nop
 1050: f3 0f 1e fa             endbr64
 1054: 68 02 00 00 00          pushq  $0x2
 1059: f2 e9 c1 ff ff ff     bnd jmpq 1020 <.plt>
 105f: 90                      nop
 1060: f3 0f 1e fa             endbr64
 1064: 68 03 00 00 00          pushq  $0x3
 1069: f2 e9 b1 ff ff ff     bnd jmpq 1020 <.plt>
 106f: 90                      nop
 1070: f3 0f 1e fa             endbr64
 1074: 68 04 00 00 00          pushq  $0x4
 1079: f2 e9 a1 ff ff ff     bnd jmpq 1020 <.plt>
 107f: 90                      nop

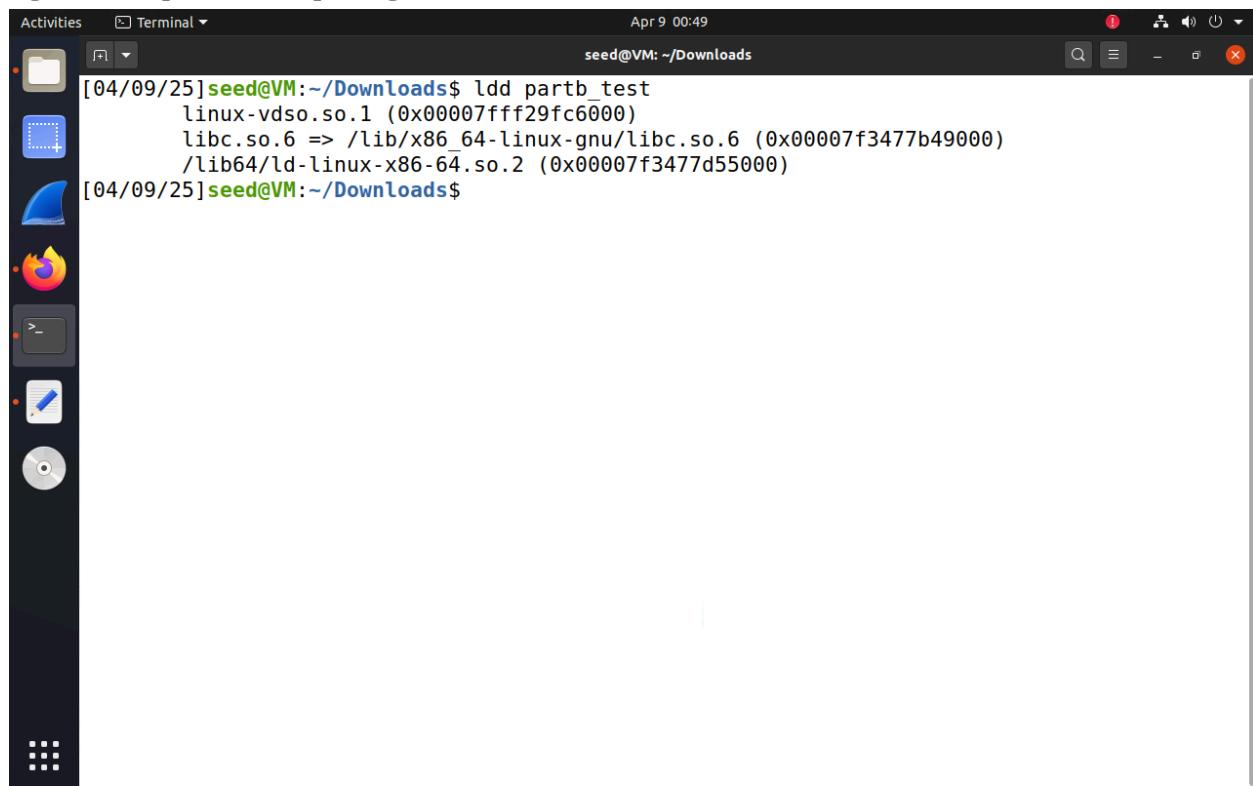
```

Figure 8: Output when inspecting Symbols



```
[04/09/25] seed@VM:~/Downloads$ nm partb_test
0000000000004010 B __bss_start
00000000000012a0 T check_password
0000000000004010 b completed.8061
0000000000004014 B counter
    w __cxa_finalize@@GLIBC_2.2.5
0000000000004000 D __data_start
0000000000004000 W data_start
0000000000001110 t deregister_tm_clones
0000000000001180 t __do_global_dtors_aux
0000000000003da0 d __do_global_dtors_aux_fini_array_entry
0000000000004008 D __dso_handle
0000000000003da8 d __DYNAMIC
0000000000004010 D __edata
0000000000004018 B __end
0000000000001358 T __fini
00000000000011c0 t frame_dummy
0000000000003d98 d __frame_dummy_init_array_entry
00000000000021fc r __FRAME_END
0000000000003f98 d __GLOBAL_OFFSET_TABLE__
    w __gmon_start__
0000000000002064 r __GNU_EH_FRAME_HDR
0000000000001000 t __init
0000000000003da0 d __init_array_end
0000000000003d98 d __init_array_start
0000000000002000 R __IO_stdin_used
    w __ITM_deregisterTMCloneTable
    w __ITM_registerTMCloneTable
[04/09/25] seed@VM:~/Downloads$ [04/09/25] seed@VM:~/Downloads$ nm /lib/libc.so.6
0000000000004018 B __end
0000000000001358 T __fini
00000000000011c0 t frame_dummy
0000000000003d98 d __frame_dummy_init_array_entry
00000000000021fc r __FRAME_END
0000000000003f98 d __GLOBAL_OFFSET_TABLE__
    w __gmon_start__
0000000000002064 r __GNU_EH_FRAME_HDR
0000000000001000 t __init
0000000000003da0 d __init_array_end
0000000000003d98 d __init_array_start
0000000000002000 R __IO_stdin_used
    w __ITM_deregisterTMCloneTable
    w __ITM_registerTMCloneTable
0000000000001350 T __libc_csu_fini
00000000000012e0 T __libc_csu_init
    U __libc_start_main@@GLIBC_2.2.5
00000000000011c9 T main
    U printf@GLIBC_2.2.5
    U puts@GLIBC_2.2.5
0000000000001140 T register_tm_clones
0000000000001289 T show_usage
    U __stack_chk_fail@@GLIBC_2.4
00000000000010e0 T start
    U strcmp@GLIBC_2.2.5
    U strcpy@GLIBC_2.2.5
0000000000004010 D __TMC_END
[04/09/25] seed@VM:~/Downloads$
```

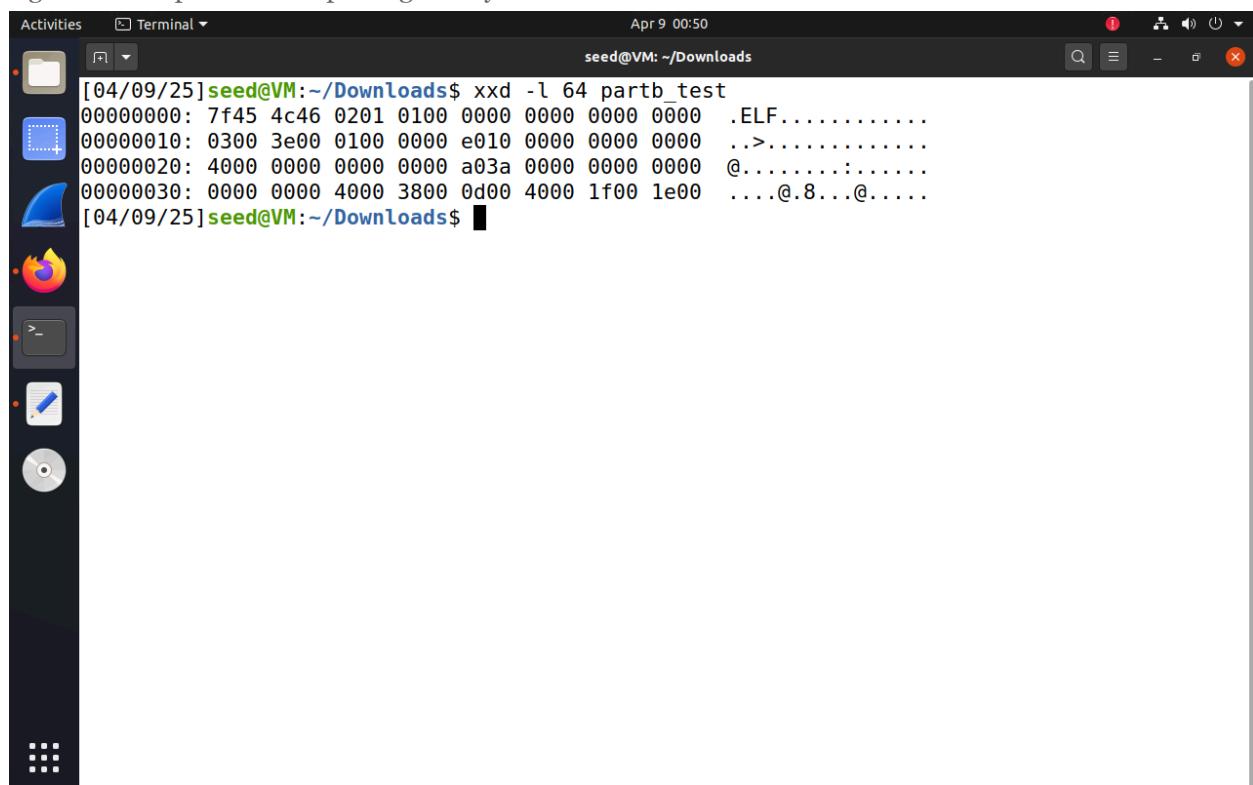
Figure 9: Output when inspecting Shared Libraries



A screenshot of a Linux desktop environment showing a terminal window. The terminal window title is "Terminal" and the command prompt is "seed@VM: ~/Downloads\$". The date and time at the top right are "Apr 9 00:49". The terminal displays the following output:

```
[04/09/25] seed@VM:~/Downloads$ ldd partb_test
    linux-vdso.so.1 (0x00007fff29fc6000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3477b49000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f3477d55000)
[04/09/25] seed@VM:~/Downloads$
```

Figure 10: Output when inspecting Binary Contents



A screenshot of a Linux desktop environment showing a terminal window. The terminal window title is "Terminal" and the command prompt is "seed@VM: ~/Downloads\$". The date and time at the top right are "Apr 9 00:50". The terminal displays the following output:

```
[04/09/25] seed@VM:~/Downloads$ xxd -l 64 partb_test
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000 .ELF.....
00000010: 0300 3e00 0100 0000 e010 0000 0000 0000 ..>.....
00000020: 4000 0000 0000 0000 a03a 0000 0000 0000 @.....:.....
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00 ....@.8...@.....
[04/09/25] seed@VM:~/Downloads$
```