vishwaCTF

CHALLENGE NAME : STOP ME IF YOU CAN

DEV : ANKUSH KAUDI

CATEGORY : REVERSE ENGINEERING

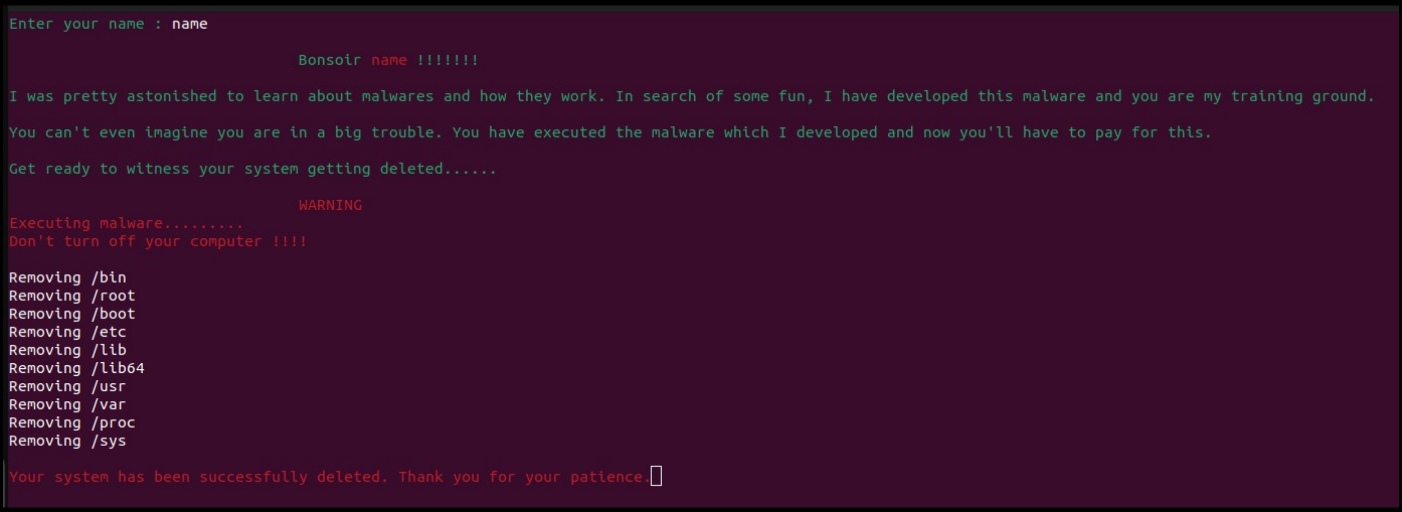
LEVEL : MEDIUM

2024

DESCRIPTION : Malwares are unstoppable. I have one such malware for you and it is pretty uncontrollable. Run it and you'll find it yourself

ATTACHMENT : malware

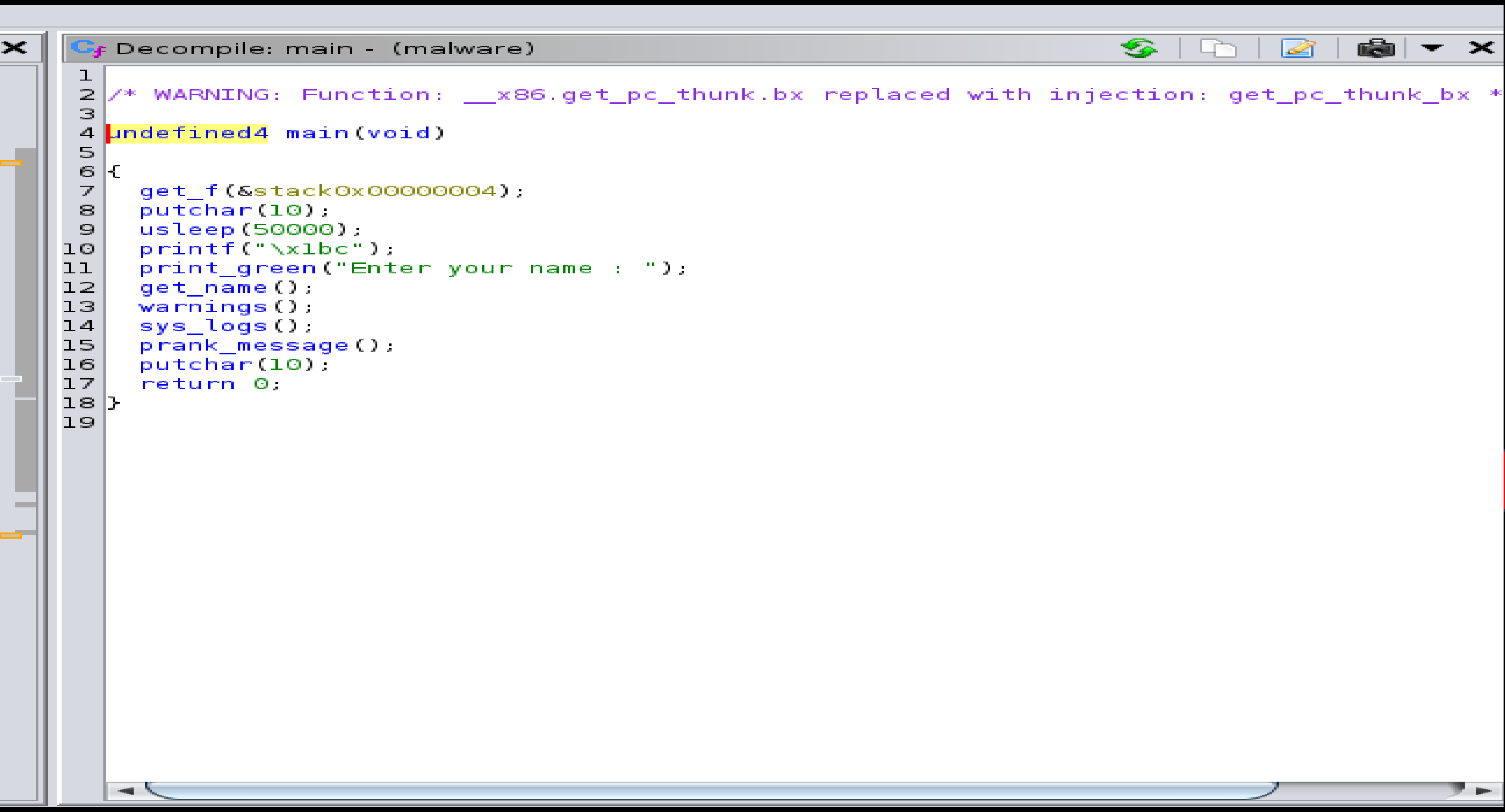
SOLUTION : Given file is a 32-bit ELF executible non-stripped binary for linux. Running it shows the following output



```
Enter your name : name

                                Bonsoir name !!!!!!!

I was pretty astonished to learn about malwares and how they work. In search of some fun, I have developed this malware and you are my training ground.

You can't even imagine you are in a big trouble. You have executed the malware which I developed and now you'll have to pay for this.

Get ready to witness your system getting deleted......

                                WARNING
Executing malware.........
Don't turn off your computer !!!!

Removing /bin
Removing /root
Removing /boot
Removing /etc
Removing /lib
Removing /lib64
Removing /usr
Removing /var
Removing /proc
Removing /sys

Your system has been successfully deleted. Thank you for your patience.
```

If we notice, when the executible is run flag flashes for very short time such that it cannot be read easily. To solve this challenge, first we need to analyse how the flag is being printed. Using ghidra to analyse the binary, we can find the following

Analysing the main function, we can see a call to get_f() function



```
/* WARNING: Function: __x86.get_pc_thunk.bx replaced with injection: get_pc_thunk_bx *

undefined4 main(void)

{
  get_f(&stack0x00000004);
  putchar(10);
  usleep(50000);
  printf("\x1bc");
  print_green("Enter your name : ");
  get_name();
  warnings();
  sys_logs();
  prank_message();
  putchar(10);
  return 0;
}
```

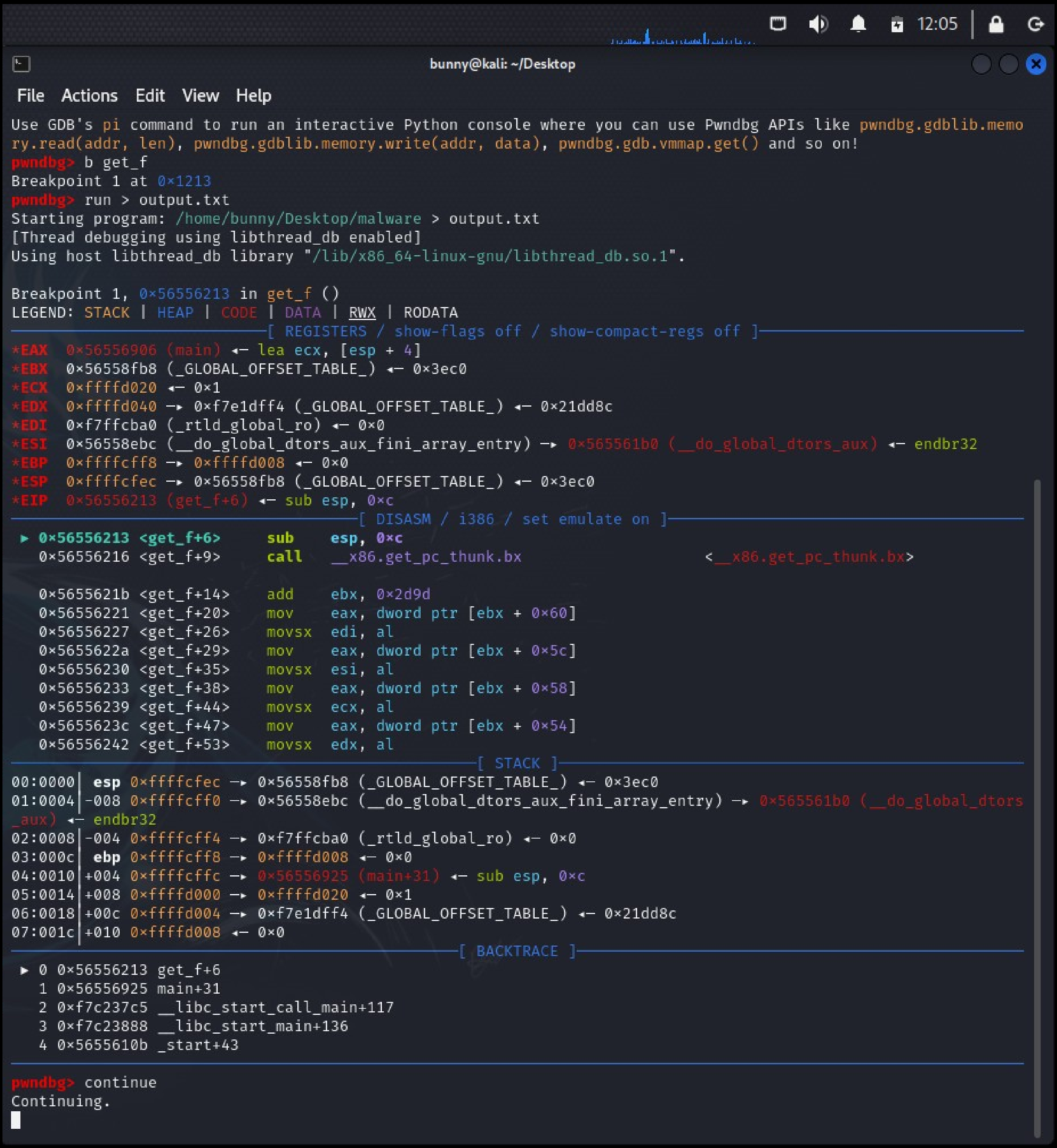
Analysing the get_f() function we can see printf() statements printing char values of intergers.

We can use GDB debugger to capture the console output and store it in a text file which can be done as following :

1. Start GDB with the binary as input

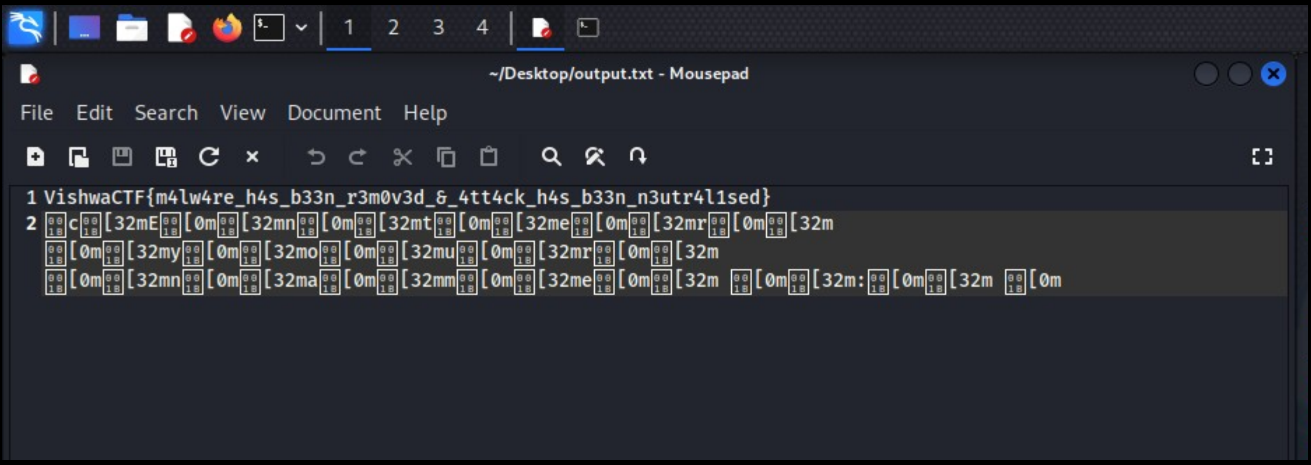
2. Set breakpoint at get_f() function

3. Use run command and store it in text file



```
Use GDB's pi command to run an interactive Python console where you can use Pwndbg APIs like pwndbg.gdblib.memo
ry.read(addr, len), pwndbg.gdblib.memory.write(addr, data), pwndbg.gdb.vmmap.get() and so on!
pwndbg> b get_f
Breakpoint 1 at 0x1213
pwndbg> run > output.txt
Starting program: /home/bunny/Desktop/malware > output.txt
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x56556213 in get_f ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
───────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]───────────────────────────────
*EAX  0x56556906 (main) ◂— lea ecx, [esp + 4]
*EBX  0x56558fb8 (_GLOBAL_OFFSET_TABLE_) ◂— 0x3ec0
*ECX  0xffffd020 ◂— 0x1
*EDX  0xffffd040 —▸ 0xf7e1dff4 (_GLOBAL_OFFSET_TABLE_) ◂— 0x21dd8c
*EDI  0xf7ffcba0 (_rtld_global_ro) ◂— 0x0
*ESI  0x56558ebc (__do_global_dtors_aux_fini_array_entry) —▸ 0x565561b0 (__do_global_dtors_aux) ◂— endbr32
*EBP  0xffffcff8 —▸ 0xffffd008 ◂— 0x0
*ESP  0xffffcfec —▸ 0x56558fb8 (_GLOBAL_OFFSET_TABLE_) ◂— 0x3ec0
*EIP  0x56556213 (get_f+6) ◂— sub esp, 0xc
──────────────────────────────────────[ DISASM / i386 / set emulate on ]──────────────────────────────────────
 ▶ 0x56556213 <get_f+6>     sub    esp, 0xc
   0x56556216 <get_f+9>     call   __x86.get_pc_thunk.bx                      <__x86.get_pc_thunk.bx>

   0x5655621b <get_f+14>    add    ebx, 0x2d9d
   0x56556221 <get_f+20>    mov    eax, dword ptr [ebx + 0x60]
   0x56556227 <get_f+26>    movsx  edi, al
   0x5655622a <get_f+29>    mov    eax, dword ptr [ebx + 0x5c]
   0x56556230 <get_f+35>    movsx  esi, al
   0x56556233 <get_f+38>    mov    eax, dword ptr [ebx + 0x58]
   0x56556239 <get_f+44>    movsx  ecx, al
   0x5655623c <get_f+47>    mov    eax, dword ptr [ebx + 0x54]
   0x56556242 <get_f+53>    movsx  edx, al
───────────────────────────────────────────────[ STACK ]───────────────────────────────────────────────
00:0000│ esp 0xffffcfec —▸ 0x56558fb8 (_GLOBAL_OFFSET_TABLE_) ◂— 0x3ec0
01:0004│-008 0xffffcff0 —▸ 0x56558ebc (__do_global_dtors_aux_fini_array_entry) —▸ 0x565561b0 (__do_global_dtors
_aux) ◂— endbr32
02:0008│-004 0xffffcff4 —▸ 0xf7ffcba0 (_rtld_global_ro) ◂— 0x0
03:000c│ ebp 0xffffcff8 —▸ 0xffffd008 ◂— 0x0
04:0010│+004 0xffffcffc —▸ 0x56556925 (main+31) ◂— sub esp, 0xc
05:0014│+008 0xffffd000 —▸ 0xffffd020 ◂— 0x1
06:0018│+00c 0xffffd004 —▸ 0xf7e1dff4 (_GLOBAL_OFFSET_TABLE_) ◂— 0x21dd8c
07:001c│+010 0xffffd008 ◂— 0x0
─────────────────────────────────────────────[ BACKTRACE ]─────────────────────────────────────────────
 ▶ 0 0x56556213 get_f+6
   1 0x56556925 main+31
   2 0xf7c237c5 __libc_start_call_main+117
   3 0xf7c23888 __libc_start_main+136
   4 0x5655610b _start+43
───────────────────────────────────────────────────────────────────────────────────────────────────────

pwndbg> continue
Continuing.
```

From above process, the console output will be stored in output.txt. The contents of the output.txt is as follows :



```
VishwaCTF{m4lw4re_h4s_b33n_r3m0v3d_&_4tt4ck_h4s_b33n_n3utr4l1sed}
␛c␛[32mE␛[0m␛[32mn␛[0m␛[32mt␛[0m␛[32me␛[0m␛[32mr␛[0m␛[32m
␛[0m␛[32my␛[0m␛[32mo␛[0m␛[32mu␛[0m␛[32mr␛[0m␛[32m
␛[0m␛[32mn␛[0m␛[32ma␛[0m␛[32mm␛[0m␛[32me␛[0m␛[32m ␛[0m␛[32m:␛[0m␛[32m ␛[0m
```

We can see the flag has been captured

Flag : VishwaCTF{m4lw4re_h4s_b33n_r3m0v3d_&_4tt4ck_h4s_b33n_n3utr4l1sed}