

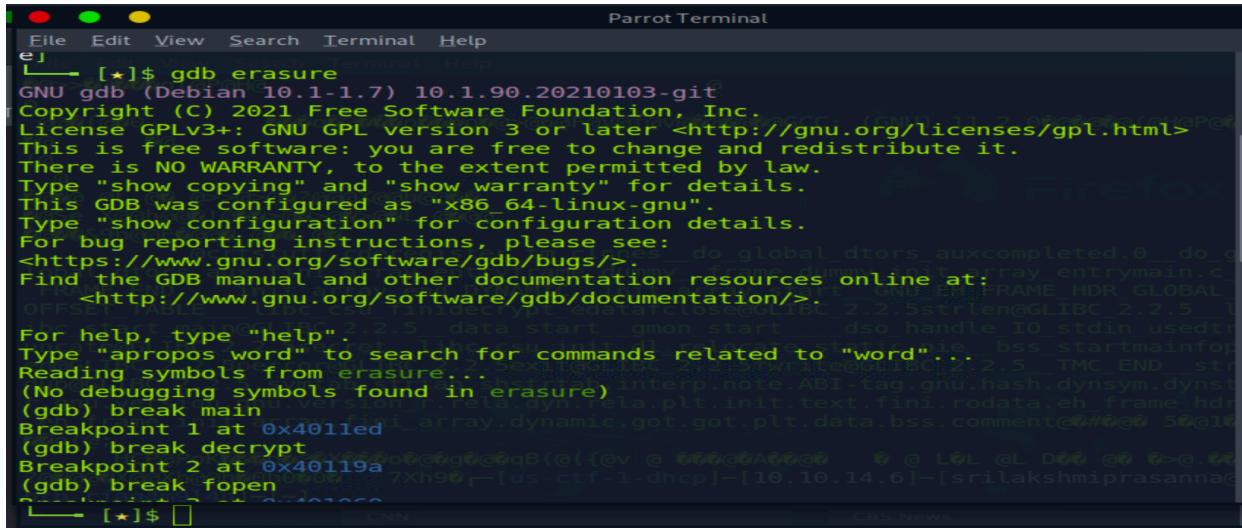
# Erasures Challenge

Step1: Downloaded the zip file and unzipped then checked the contents of the file and found it is ELF file. Then tried to execute the erasure file normally as ELF file which gave nothing as shown below.

Step2: From description understood, I understood that there might be a function that could be deleting/erasing the output. So I inspected the ELF file content and found words like exit, fopen, perror, fclose, fwrite, main and decrypt and wflag.txt as shown in the below screenshots.

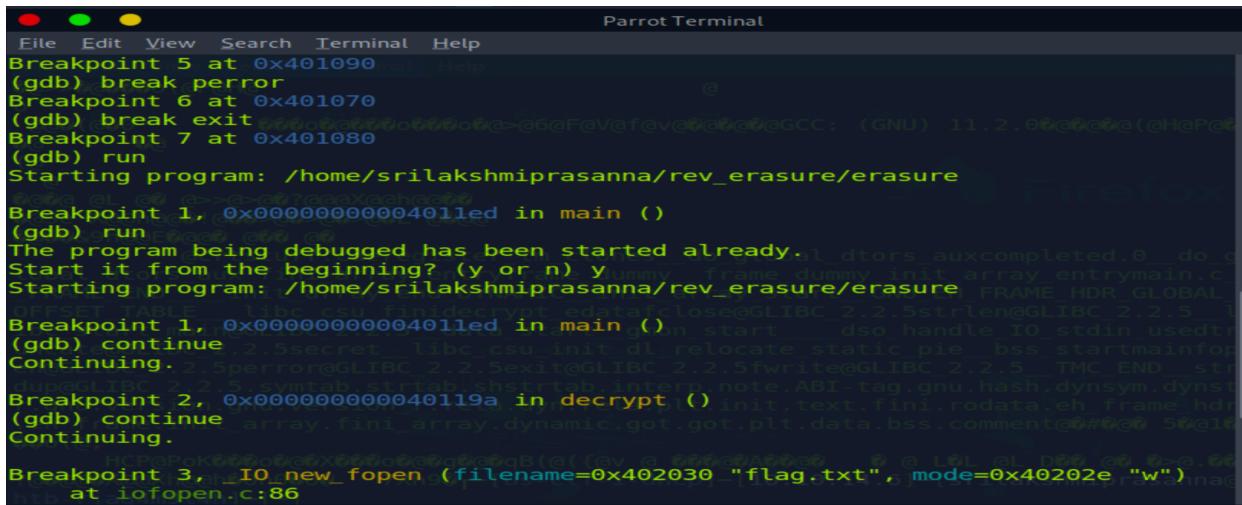
Step3: Then thought they could be functions names in the whole code. So I used GDB(GNB Debugger) which could break the code files at certain points

such as line by line or function by function and then executes. Hence, I broke the code using the functions names found above.



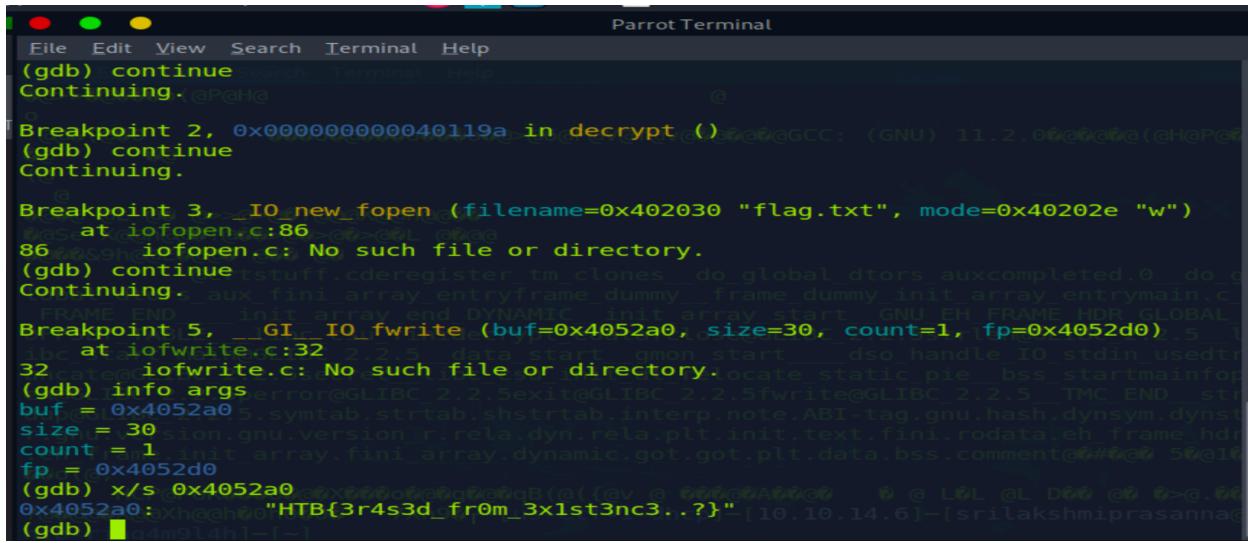
```
Parrot Terminal
File Edit View Search Terminal Help
[★]$ gdb erasure
GNU gdb (Debian 10.1.1-7) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
Reading symbols from erasure...done.
(No debugging symbols found in erasure)
(gdb) break main
Breakpoint 1 at 0x4011ed
(gdb) break decrypt
Breakpoint 2 at 0x40119a
(gdb) break fopen
[★]$
```

Step4: Then started running code from one function to another functions and found flag.txt as shown in the below screenshot.



```
Parrot Terminal
File Edit View Search Terminal Help
Breakpoint 5 at 0x401090 in main
Breakpoint 6 at 0x401070 in exit
Breakpoint 7 at 0x401080 in run
Starting program: /home/srilakshmiprasanna/rev_erasure/erasure
Breakpoint 1, 0x0000000004011ed in main ()
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n): y
Starting program: /home/srilakshmiprasanna/rev_erasure/erasure
Breakpoint 1, 0x0000000004011ed in main ()
(gdb) continue
Continuing.
Breakpoint 2, 0x00000000040119a in decrypt ()
(gdb) continue
Continuing.
Breakpoint 3, 0x000000000402030 in IO_new_fopen (filename=0x402030 "flag.txt", mode=0x40202e "w")
at iofopen.c:86
```

Step5: I still continued the execution and got the contents of the flag.txt file with help of ‘fwrite’ function which gave buffer(contains data to be written by ‘fwrite’ function) and accessed buffer and got flag as shown in the below screenshot.



The screenshot shows a terminal window titled "Parrot Terminal". The terminal is running a GDB session on a program named "decrypt". The user has set three breakpoints: Breakpoint 2 at address 0x000000000040119a, Breakpoint 3 at address 0x402030, and Breakpoint 5 at address 0x4052a0. When the program reaches each breakpoint, it prints the assembly code for that specific instruction. For example, at Breakpoint 3, it prints the assembly for the fopen call, which fails because "flag.txt" does not exist. The user continues the program after each breakpoint to observe the behavior.

```
(gdb) continue
Continuing.

Breakpoint 2, 0x000000000040119a in decrypt ()
(gdb) continue
Continuing.

Breakpoint 3, _IO_new_fopen (filename=0x402030 "flag.txt", mode=0x40202e "w")
at iofopen.c:86
86     iofopen.c: No such file or directory.
(gdb) continue
Continuing.

Breakpoint 5, __GI__IO_fwrite (buf=0x4052a0, size=30, count=1, fp=0x4052d0)
at iofwrite.c:32
32     iofwrite.c: No such file or directory.
(gdb) info args
error@GLIBC_2.2.5:exit@GLIBC_2.2.5:fwrite@GLIBC_2.2.5:TMC_END_st
buf = 0x4052a0
size = 30
sion.gnu.version_r.rela.dyn.rela.plt.init.text.fini.rodata.eh.frame.hdi
count = 1
init.array.fini.array.dynamic.got.got=plt.data.bss.comment@#0@0
fp = 0x4052d0
(gdb) x/s 0x4052a0
0x4052a0:xh@ch00[HTB{3r4s3d_froM_3x1st3nc3..?}]-[10.10.14.6]-(srilakshmiprasannad
(gdb) 
```

## Risk mitigation strategy:

This program might be created with code to erase the information to stop others seeing the sensitive data. But with the help of GDB, I was able to disclose the sensitive data.

We can mitigate this problem by providing least privileges for file access. If the file don't have 'read' and 'execute' permission for others, then GDB also might not be helpful to execute the code and access the 'buffer' of the code to see the content of the flag.txt file.

Hence least privilege principle can be good risk mitigation for information disclosure vulnerability.