

Week1- Write-up

Vishnu Vardhan Ciripuram

N14912012

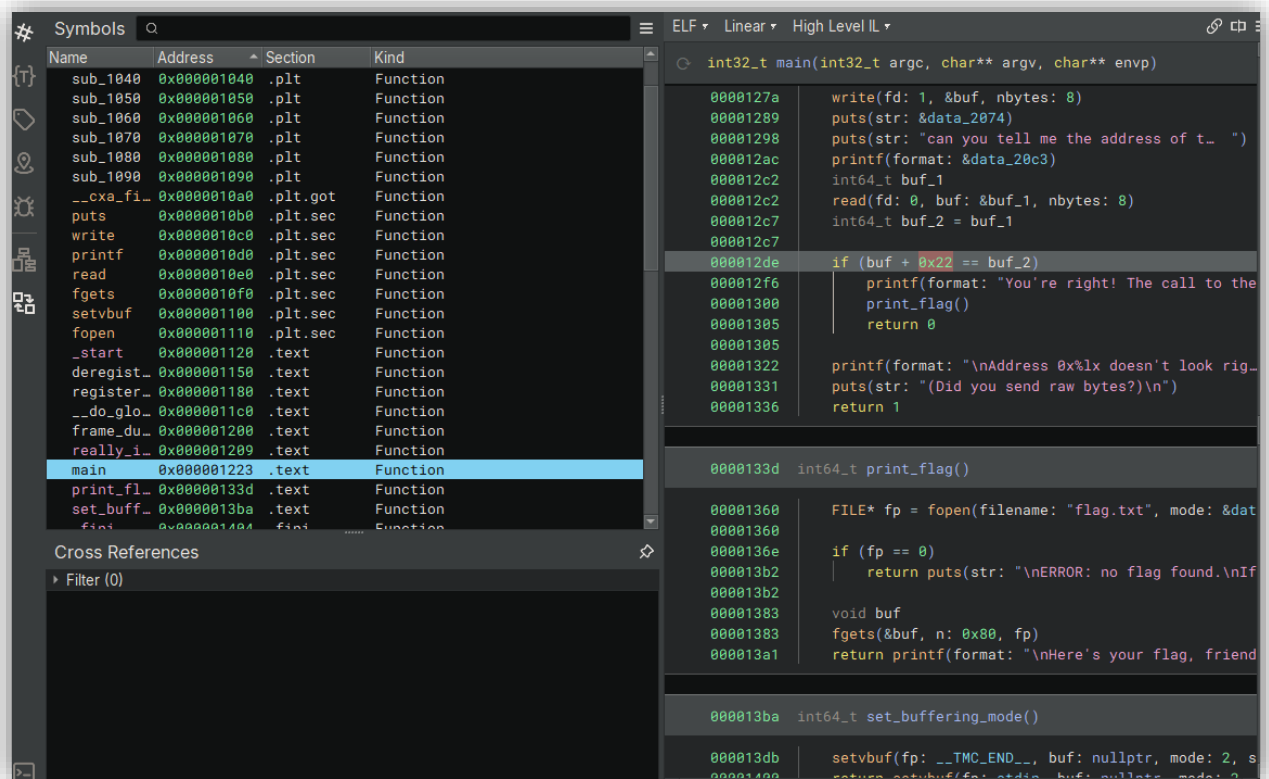
vc2499

Challenge – Directions

we are given the raw bytes address of the main() code and asked to figure out where the "really_important_function" is .

Solution steps -

- I used **pwntools** to connect to the server and sent my NetID (vc2499).
- The server sent a 6-byte raw address of main(), which I converted to a 64-bit integer using `u64()`.
- I got that the value 0x22 which is added to main() by putting the given file into **Binary Ninja** tool .



- I added the value to the main() address to calculate the really_important_function address.
- I packed the calculated address into 6 bytes and sent it to the server.
- After sending the address, I used `p.interactive()` to get the flag.
- Submission -

flag{st4t1c_4n4lys1s_g1v3s_us_s0_much_1nf0_4b0ut_4_b1n4ry!_8c85bdb794f4ab68}

```

1 from pwn import *
2
3 conn = remote('offsec-chalbroker.osiris.cyber.nyu.edu', 1244)
4
5 print(conn.recvuntil(b'NetID (something like abc123): ').decode())
6 conn.sendline(b'vc2499')
7
8 print(conn.recvuntil(b'I found the raw bytes address of main() written somewhere: ').decode())
9 main_addr = u64(conn.recv(6).ljust(8, b'\x00'))
10
11 func_addr = main_addr + 0x22
12 conn.send(p64(func_addr)[:6])
13
14 conn.interactive()
15

```

```

(kali㉿kali)-[~/Downloads/offsec]
$ python3 directions.py
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1244: Done
[*] Switching to interactive mode
\x00\x00
can you tell me the address of the call to the really_important_function?
> You're right! The call to the really_important_function is at 0x562b23a46245!

Here's your flag, friend: flag{st4t1c_4n4lys1s_g1v3s_us_s0_much_1nf0_4b0ut_4_b1n4ry!_8c85bdb794f4ab68}

[*] Got EOF while reading in interactive
$ 

```

Challenge – GDB0

Challenge is to find the password by debugging the binary using GDB

Solution steps -

- I first set a breakpoint at the main() function to understand the flow of the program using
 - “break main”
 - “run”

```

pwndbg> break main
Breakpoint 1 at 0x55a3c81bb245: file gdb0.c, line 19.
pwndbg> run
Starting program: /home/ctf/gdb0/gdb0
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7ffd41e170f8) at gdb0.c:19
19      set_buffering_mode();
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
RAX 0x5647e0e49229 (main) ← endbr64
RBX 0
RCX 0x5647e0e4bd88 (__do_global_dtors_aux_fini_array_entry) → 0x5647e0e491e0 (__do_global_dtors_aux) ←
endbr64
RDX 0x7ffd41e17108 → 0x7ffd41e18e7a ← 'LANGUAGE=en_US:en'
RDI 1
RSI 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
R8 0x7f6d29a3ef10 (initial+16) ← 4
R9 0x7f6d29a58040 (_dl_fini) ← endbr64
R10 0x7f6d29a52908 ← 0xd00120000000e
R11 0x7f6d29a6d660 (_dl_audit_preinit) ← endbr64
R12 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
R13 0x5647e0e49229 (main) ← endbr64
R14 0x5647e0e4bd88 (__do_global_dtors_aux_fini_array_entry) → 0x5647e0e491e0 (__do_global_dtors_aux) ←
endbr64
R15 0x7f6d29a8c040 (_rtld_global) → 0x7f6d29a8d2e0 → 0x5647e0e48000 ← 0x10102464c457f
RBP 0x7ffd41e16fe0 ← 1
RSP 0x7ffd41e16f30 → 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
RIP 0x5647e0e49245 (main+28) ← mov eax, 0
[ STACK ]
00:0000 | rsp 0x7ffd41e16f30 → 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
01:0008 | -0a8 0x7ffd41e16f38 ← 0x100000000
02:0010 | -0a0 0x7ffd41e16f40 ← 0
... ↓ 5 skipped
[ SOURCE (CODE) ]
In file: /home/ctf/gdb0.c:19
15 int main(int argc, char** argv) {
16     char flag[0x80];
17     char buffer[0x20];
18
19     set_buffering_mode();
20     puts("\n\n\tHEEEEEELP! My password is somewhere around here, but I can't find it.");
21     puts("\tCan you tell me my password?");
22
23     printf("\t> ");
[ DISASM / x86-64 / set emulate on ]
▶ 0x5647e0e49245 <main+28> mov     eax, 0
0x5647e0e4924a <main+33> call   set_buffering_mode    <set_buffering_mode>

0x5647e0e4924f <main+38> lea     rax, [rip + 0xdb2]
0x5647e0e49256 <main+45> mov     rdi, rax
0x5647e0e49259 <main+48> call   puts@plt             <puts@plt>

0x5647e0e4925e <main+53> lea     rax, [rip + 0xdea]
0x5647e0e49265 <main+60> mov     rdi, rax
0x5647e0e49268 <main+63> call   puts@plt             <puts@plt>

0x5647e0e4926d <main+68> lea     rax, [rip + 0xdf9]
0x5647e0e49274 <main+75> mov     rdi, rax
0x5647e0e49277 <main+78> mov     eax, 0
pwndbg>

```

- I stepped through the program to understand how it processes the password input. I identified that the program calls the `get_password()` function to retrieve the correct password for comparison.
- The code revealed that the program reads user input with `fgets()` and compares the input with the return value of `get_password()`.

```

pwndbg> next
> 24      fgets(buffer, sizeof(buffer), stdin);
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
*RAX 3
RBX 0
RCX 0x7f6d29937887 (write+23) ← cmp rax, -0x1000 /* 'H=' */
*RDY 0
*RDI 0x7ffd41e14cf0 → 0x7f6d29885050 (funlockfile) ← endbr64
*RSI 0x7ffd41e14e10 ← 0x203e09 /* '\t> ' */
*R8 3
R9 0x7f6d29a58040 (_dl_fini) ← endbr64
*R10 0x5647e0e4a06d ← 0xa00203e09 /* '\t> ' */
R11 0x246
R12 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
R13 0x5647e0e49229 (main) ← endbr64
R14 0x5647e0e4bd88 (__do_global_ctors_aux_fini_array_entry) → 0x5647e0e491e0 (__do_global_ctors_aux) ← endbr64
R15 0x7f6d29a8c040 (_rtld_global) → 0x7f6d29a8d2e0 → 0x5647e0e48000 ← 0x10102464c457f
RBP 0x7ffd41e16fe0 ← 1
RSP 0x7ffd41e16f30 → 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
*RIP 0x5647e0e49281 (main+88) ← mov rdx, qword ptr [rip + 0x2db8]
[ STACK ]
00:0000 | rsp 0x7ffd41e16f30 → 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
01:0008 | -0a8 0x7ffd41e16f38 ← 0x100000000
02:0010 | -0a0 0x7ffd41e16f40 ← 0
... ↓ 5 skipped
[ SOURCE (CODE) ]
In file: /home/ctf/gdb0.c:24
20 puts("\n\n\tHEEEELP! My password is somewhere around here, but I can't find it.");
21 puts("\tCan you tell me my password?");
22
23 printf("\t> ");
24 ► fgets(buffer, sizeof(buffer), stdin);
25 buffer[strcspn(buffer, "\n")] = '\0';
26
27 if (strcmp(buffer, get_password()) == 0) {
28     puts("\tYou did it! You found my password!");
}
[ DISASM / x86-64 / set emulate on ]
0x5647e0e49268 <main+63> call puts@plt <puts@plt>

0x5647e0e4926d <main+68> lea rax, [rip + 0xdf9]
0x5647e0e49274 <main+75> mov rdi, rax
0x5647e0e49277 <main+78> mov eax, 0
0x5647e0e4927c <main+83> call printf@plt <printf@plt>

► 0x5647e0e49281 <main+88> mov rdx, qword ptr [rip + 0x2db8]
0x5647e0e49288 <main+95> lea rax, [rbp - 0xa0]
0x5647e0e4928f <main+102> mov esi, 0x20
0x5647e0e49294 <main+107> mov rdi, rax
0x5647e0e49297 <main+110> call fgets@plt <fgets@plt>

0x5647e0e4929c <main+115> lea rax, [rbp - 0xa0]
pwndbg>

```

- Since the `get_password()` function holds the actual password, I set a breakpoint at this function to inspect the return value.
 - `break get_password`
 - `continue`

```

pwndbg> break get_password
Breakpoint 2 at 0x5647e0e4933b: file gdb0.c, line 41.
pwndbg> continue
Continuing.
next

Breakpoint 2, get_password () at gdb0.c:41
41      return password;
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
*RAX 0
*RBX 0
*RCX 4
*RDX 1
*RDI 0x7ffd41e16f40 ← 0x7478656e /* 'next' */
*RSI 1
*R8 0x5647e0e4a070 ← 0xa00
*R9 0
R10 0x5647e0e4a06d ← 0xa00203e09 /* '\t> ' */
R11 0x246
R12 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
R13 0x5647e0e49229 (main) ← endbr64
R14 0x5647e0e4bd88 (__do_global_dtors_aux_fini_array_entry) → 0x5647e0e491e0 (__do_global_dtors_aux) ← endbr64
R15 0x7f6d29a8c040 (_rtld_global) → 0x7f6d29a8d2e0 → 0x5647e0e48000 ← 0x10102464c457f
*RBP 0x7ffd41e16f20 → 0x7ffd41e16fe0 ← 1
*RSP 0x7ffd41e16f20 → 0x7ffd41e16fe0 ← 1
*RIP 0x5647e0e4933b (get_password+8) ← lea rax, [rip + 0x2cce]
[ STACK ]
00:0000 | rbp rsp 0x7ffd41e16f20 → 0x7ffd41e16fe0 ← 1
01:0008 | +008 0x7ffd41e16f28 → 0x5647e0e492c7 (main+158) ← mov rdx, rax
02:0010 | +010 0x7ffd41e16f30 → 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
03:0018 | +018 0x7ffd41e16f38 ← 0x100000000
04:0020 | rdi 0x7ffd41e16f40 ← 0x7478656e /* 'next' */
05:0028 | +028 0x7ffd41e16f48 ← 0
... ↓ 2 skipped
[ SOURCE (CODE) ]
In file: /home/ctf/gdb0.c:41
37 }
38
39
40 char* get_password() {
▶ 41     return password;
42 }
43
44
45 void set_buffering_mode() {
[ DISASM / x86-64 / set emulate on ]
▶ 0x5647e0e4933b <get_password+8> lea rax, [rip + 0x2cce]
0x5647e0e49342 <get_password+15> pop rbp
0x5647e0e49343 <get_password+16> ret <main+158>
↓
0x5647e0e492c7 <main+158> mov rdx, rax
0x5647e0e492ca <main+161> lea rax, [rbp - 0xa0]
0x5647e0e492d1 <main+168> mov rsi, rdx
0x5647e0e492d4 <main+171> mov rdi, rax
0x5647e0e492d7 <main+174> call strcmp@plt <strcmp@plt>
0x5647e0e492dc <main+179> test eax, eax
0x5647e0e492de <main+181> jne main+244 <main+244>
0x5647e0e492e0 <main+183> lea rax, [rip + 0xd91]
pwndbg>

```

- When I hit the breakpoint at `get_password()`, I stepped through the function using “next” until the RAX register (which holds the return value) contained the address of the password.

```

pwndbg> next
42
}
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
-----[ REGISTERS / show-flags off / show-compact-regs off ]-----
*RAX 0x5647e0e4c010 (password) ← '4_v3ry_1337_s3cr3t_p4ssw0rd'
RBX 0
RCX 4
RDX 1
RDI 0x7ffd41e16f40 ← 0x7478656e /* 'next' */
RSI 1
R8 0x5647e0e4a070 ← 0xa00
R9 0
R10 0x5647e0e4a06d ← 0xa00203e09 /* '\t> ' */
R11 0x246
R12 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
R13 0x5647e0e49229 (main) ← endbr64
R14 0x5647e0e4bd88 (__do_global_dtors_aux_fini_array_entry) → 0x5647e0e491e0 (__do_global_dtors_aux) ←
- endbr64
R15 0x7f6d29a8c040 (_rtld_global) → 0x7f6d29a8d2e0 → 0x5647e0e48000 ← 0x10102464c457f
RBP 0x7ffd41e16f20 → 0x7ffd41e16fe0 ← 1
RSP 0x7ffd41e16f20 → 0x7ffd41e16fe0 ← 1
*RIP 0x5647e0e49342 (get_password+15) ← pop rbp
-----[ STACK ]-----
00:0000 | rbp rsp 0x7ffd41e16f20 → 0x7ffd41e16fe0 ← 1
01:0008 | +008 0x7ffd41e16f28 → 0x5647e0e492c7 (main+158) ← mov rdx, rax
02:0010 | +010 0x7ffd41e16f30 → 0x7ffd41e170f8 → 0x7ffd41e18e6b ← '/home/ctf/gdb0'
03:0018 | +018 0x7ffd41e16f38 ← 0x100000000
04:0020 | rdi 0x7ffd41e16f40 ← 0x7478656e /* 'next' */
05:0028 | +028 0x7ffd41e16f48 ← 0
... ↓ 2 skipped
-----[ SOURCE (CODE) ]-----
In file: /home/ctf/gdb0.c:42
38
39
40 char* get_password() {
41     return password;
42 }
43
44
45 void set_buffering_mode() {
46     setvbuf(stdout, NULL, _IONBF, 0);
-----[ DISASM / x86-64 / set emulate on ]-----
0x5647e0e4933b <get_password+8> lea rax, [rip + 0x2cce]
▶ 0x5647e0e49342 <get_password+15> pop rbp
0x5647e0e49343 <get_password+16> ret <main+158>
↓
0x5647e0e492c7 <main+158> mov rdx, rax
0x5647e0e492ca <main+161> lea rax, [rbp - 0xa0]
0x5647e0e492d1 <main+168> mov rsi, rdx
0x5647e0e492d4 <main+171> mov rdi, rax
0x5647e0e492d7 <main+174> call strcmp@plt <strcmp@plt>
0x5647e0e492dc <main+179> test eax, eax
0x5647e0e492de <main+181> jne main+244 <main+244>
0x5647e0e492e0 <main+183> lea rax, [rip + 0xd91]
pwndbg>

```

- I used “x/s \$rax” to get the password:
- The password was revealed to be: 4_v3ry_1337_s3cr3t_p4ssw0rd

```

pwndbg> x/s $rax
0x5647e0e4c010 <password>: "4_v3ry_1337_s3cr3t_p4ssw0rd"
pwndbg>

```


- After retrieving the password, I entered it when prompted by the program, which resulted in successfully solving the challenge and receiving the flag.

flag{34sy_3n0ugh_wh3n_y0u_g3t_d3bug_symb0ls!_d186793f0f8cb398}

```

kali@kali: ~
File Actions Edit View Help

[crash] Welcome to GDB 0

This time you will have access to the source code!

File System
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

HEEEEELP! My password is somewhere around here, but I can't find it.
Can you tell me my password?
> 4_v3ry_1337_s3cr3t_p4ssw0rd
You did it! You found my password!

Here's your flag, friend: flag{34sy_3n0ugh_wh3n_y0u_g3t_d3bug_symb0ls!_d186793f0f8cb398}

[Inferior 1 (process 66) exited normally]

Remember, you can type the command `run` to run the program
or type `break main` and then `run` to start debugging!

pwndbg>

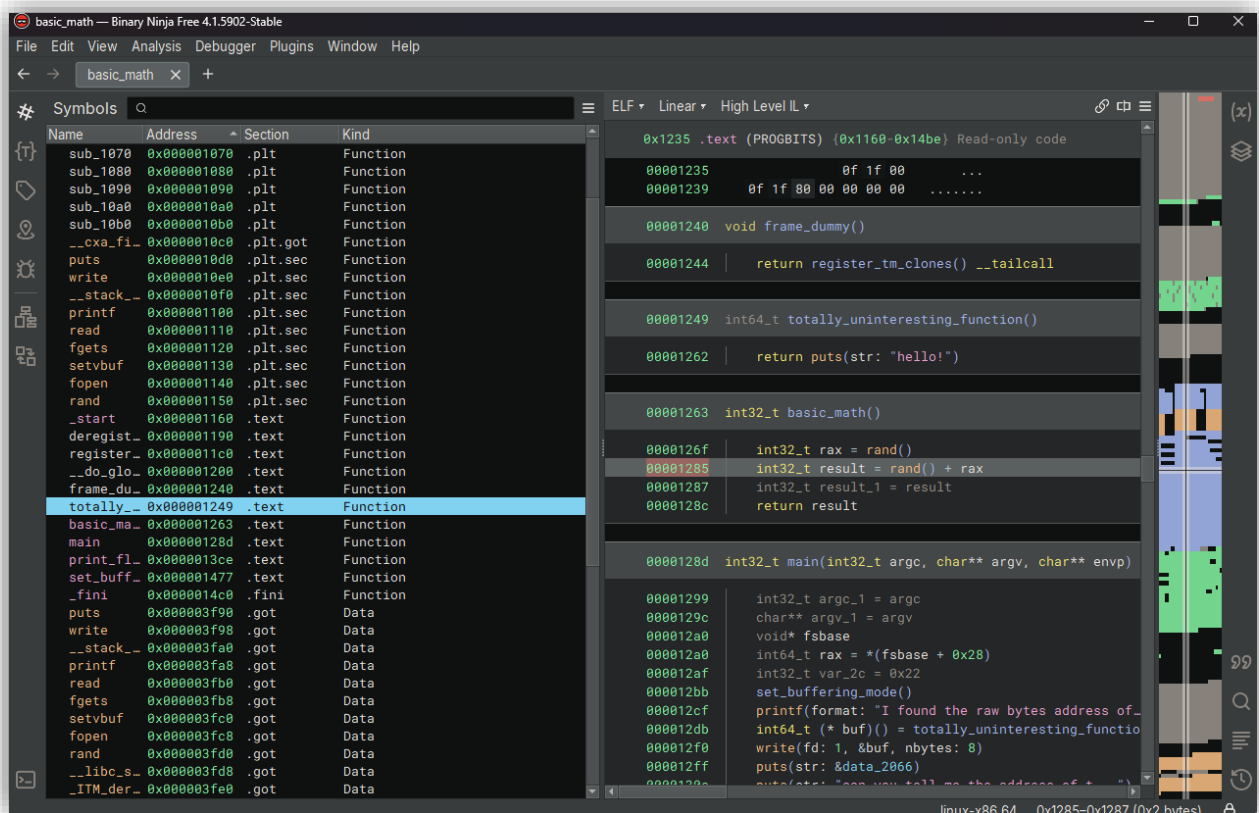
```

Challenge – Basic Math

We are given the raw bytes address of the `totally_uninteresting_function` and are asked to figure out where the `ADD` instruction is located in the `basic_math` function.

Solution Steps:

- I used **pwntools** to connect to the server and sent my NetID (vc2499).
- The server sent a 6-byte raw address of the `totally_uninteresting_function`, which I converted to a 64-bit integer using `u64()`.
- I analyzed the given binary using **Binary Ninja**, where I found:
 - The offset of `totally_uninteresting_function` is `0x1249`.
 - The offset of the `ADD` instruction in `basic_math` is `0x1285`.



- To calculate the base address, I subtracted the offset 0x1249 from the raw address of `totally_uninteresting_function` that the server provided.
- I then added the offset 0x1285 to the base address to compute the exact memory address of the ADD instruction.
- I packed the calculated ADD instruction address into 6 bytes and sent it to the server.
- After sending the correct address, I used `p.interactive()` to interact with the server and retrieve the flag.

flag{R34d1ng_4ss3mbly_l4ngu4ge_w4snt_th4t_h4rd!_ac16eabc5199b51b}


```
Open basicMath.py Save
~/Downloads/offsec

1 from pwn import *
2
3 HOST = "offsec-chalbroker.osiris.cyber.nyu.edu"
4 PORT = 1245
5 NETID = b"vc2499"
6
7 TOTALLY_UNINTERESTING_FUNC_OFFSET = 0x1249 # Offset of totally_uninteresting_function
8 ADD_INSTRUCTION_OFFSET = 0x1285 # Offset of the ADD instruction in basic_math
9
10 p = remote(HOST, PORT)
11 p.recvuntil(b"NetID (something like abc123): ")
12 p.sendline(NETID)
13
14 p.recvuntil(b'I found the raw bytes address of `totally_uninteresting_function` written somewhere: ')
15 raw_func_addr = p.recv(6)
16
17 func_addr = u64(raw_func_addr.ljust(8, b'\x00'))
18
19 base_addr = func_addr - TOTALLY_UNINTERESTING_FUNC_OFFSET
20
21 add_instruction_addr = base_addr + ADD_INSTRUCTION_OFFSET
22 raw_add_instruction_addr = p64(add_instruction_addr)[6]
23
24 p.send(raw_add_instruction_addr)
25
26 p.interactive()
27
```

```
(kali㉿kali)-[~/Downloads/offsec]
$ python3 basicMath.py
[+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 1245: Done
[*] Switching to interactive mode
\x00\x00
can you tell me the address of the ADD instruction in basic_math?
> You're right! The call to the add instruction is at 0x55c17de21285!

Here's your flag, friend: flag{R34d1ng_4ss3mbly_l4ngu4ge_w4snt_th4t_h4rd!_ac16eabc5199b51b}

[*] Got EOF while reading in interactive
$ c
```

Challenge – GDB 2

Challenge is to extract a flag from a binary called gdb2. The binary didn't require any input and the challenge hinted that the flag could be obtained through debugging.

Solution Steps:

- I began by disassembling the main() function to get a sense of how the program works.
 - disassemble main

```
pwndbg> disassemble main
Dump of assembler code for function main:
0x0000000000001209 <+0>:    endbr64
0x000000000000120d <+4>:    push    rbp
0x000000000000120e <+5>:    mov     rbp, rsp
0x0000000000001211 <+8>:    sub     rsp, 0x10
0x0000000000001215 <+12>:   mov     DWORD PTR [rbp-0x4], edi
0x0000000000001218 <+15>:   mov     QWORD PTR [rbp-0x10], rsi
0x000000000000121c <+19>:   mov     eax, 0x0
0x0000000000001221 <+24>:   call    0x12c7 <set_buffering_mode>
0x0000000000001226 <+29>:   mov     eax, 0x0
0x000000000000122b <+34>:   call    0x1250 <read_file>
0x0000000000001230 <+39>:   lea     rax, [rip+0xdd1]          # 0x2008
0x0000000000001237 <+46>:   mov     rdi, rax
0x000000000000123a <+49>:   call    0x10b0 <puts@plt>
0x000000000000123f <+54>:   mov     edi, 0x3
0x0000000000001244 <+59>:   call    0x1110 <sleep@plt>
0x0000000000001249 <+64>:   mov     eax, 0x0
0x000000000000124e <+69>:   leave
0x000000000000124f <+70>:   ret
End of assembler dump.
```

- While looking through the code, I saw that main() calls a function called read_file(). Since this sounded like it might be where the flag is being loaded, I decided to set a breakpoint at the start of read_file() to take a closer look at what it does.
 - break read_file
 - run

```

pwndbg> break read_file
Breakpoint 1 at 0x1258
pwndbg> run
Starting program: /home/ctf/gdb2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x0000556500ccf258 in read_file ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
RAX 0
RBX 0
RCX 0xc00
RDX 0xfbad008b
RDI 0x7f8608836a80 (_IO_stdfile_0_lock) ← 0
RSI 0
R8 0x7f8608835f10 (initial+16) ← 4
R9 0x7f860884f040 (_dl_fini) ← endbr64
R10 0x7f8608849908 ← 0xd001200000000e
R11 0x7f8608864660 (_dl_audit_preinit) ← endbr64
R12 0x7ffe9a0b6c18 → 0x7ffe9a0b6e6b ← '/home/ctf/gdb2'
R13 0x556500ccf209 (main) ← endbr64
R14 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
R15 0x7f8608883040 (_rtld_global) → 0x7f86088842e0 → 0x556500cce000 → 0x10102464c457f
RBP 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
RSP 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
RIP 0x556500ccf258 (read_file+8) ← sub rsp, 0x10
[ STACK ]
00:0000 | rbp | rsp | 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
01:0008 | +008 | 0x7ffe9a0b6ae8 → 0x556500ccf230 (main+39) ← lea rax, [rip + 0xdd1]
02:0010 | +010 | 0x7ffe9a0b6af0 → 0x7ffe9a0b6c18 → 0x7ffe9a0b6e6b ← '/home/ctf/gdb2'
03:0018 | +018 | 0x7ffe9a0b6af8 ← 0x100000000
04:0020 | +020 | 0x7ffe9a0b6b00 ← 1
05:0028 | +028 | 0x7ffe9a0b6b08 → 0x7f86088643d90 (__libc_start_call_main+128) ← mov edi, eax
06:0030 | +030 | 0x7ffe9a0b6b10 ← 0
07:0038 | +038 | 0x7ffe9a0b6b18 → 0x556500ccf209 (main) ← endbr64
[ DISASM / x86-64 / set emulate on ]
► 0x556500ccf258 <read_file+8> sub rsp, 0x10 RSP ⇒ 0x7ffe9a0b6ad0 (0x7ffe9a0b6ae0 - 0x10)
0x556500ccf25c <read_file+12> mov esi, 0 ESI ⇒ 0
0x556500ccf261 <read_file+17> lea rax, [rip + 0xddb] RAX ⇒ 0x556500cd0043 ← 'flag.txt'
0x556500ccf268 <read_file+24> mov rdi, rax RDI ⇒ 0x556500cd0043 ← 'flag.txt'
0x556500ccf26b <read_file+27> mov eax, 0 EAX ⇒ 0
0x556500ccf270 <read_file+32> call open@plt <open@plt>
0x556500ccf275 <read_file+37> mov dword ptr [rbp - 4], eax
0x556500ccf278 <read_file+40> cmp dword ptr [rbp - 4], -1
0x556500ccf27c <read_file+44> jne read_file+76 <read_file+76>
0x556500ccf27e <read_file+46> lea rax, [rip + 0xdcb] RAX ⇒ 0x556500cd0050 ← '\nERROR: no flag found.\nPlease message the CAs.'
0x556500ccf285 <read_file+53> mov rdi, rax

```

- Once the program hit the breakpoint, I started stepping through the instructions using next. It didn't take long to see that read_file() calls the open@plt function, trying to open a file called "flag.txt". That confirmed my suspicion that this file contains the flag.

```

pwndbg> next
0x0000556500ccf270 in read_file ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
RAX 0
RBX 0
RCX 0xc00
RDX 0xfbad008b
RDI 0x556500cd0043 ← 'flag.txt'
RSI 0
R8 0x7f8608835f10 (initial+16) ← 4
R9 0x7f860884f040 (__dl_fini) ← endbr64
R10 0x7f8608849908 ← 0xd00120000000e
R11 0x7f8608864660 (__dl_audit_preinit) ← endbr64
R12 0x7ffe9a0b6c18 → 0x7ffe9a0b6e6b ← '/home/ctf/gdb2'
R13 0x556500ccf209 (main) ← endbr64
R14 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
R15 0x7f8608883040 (__rtld_global) → 0x7f86088842e0 → 0x556500cce000 ← 0x10102464c457f
RBP 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
RSP 0x7ffe9a0b6ad0 → 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
RIP 0x556500ccf270 (read_file+32) ← call 0x556500ccf0f0
[ STACK ]
00:0000 | rsp 0x7ffe9a0b6ad0 → 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
01:0008 | -008 0x7ffe9a0b6ad8 → 0x556500ccf30b (set_buffering_mode+68) ← nop
02:0010 | rbp 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
03:0018 | +008 0x7ffe9a0b6ae8 → 0x556500ccf230 (main+39) ← lea rax, [rip + 0xdd1]
04:0020 | +010 0x7ffe9a0b6af0 → 0x7ffe9a0b6c18 → 0x7ffe9a0b6e6b ← '/home/ctf/gdb2'
05:0028 | +018 0x7ffe9a0b6af8 → 0x1000000000 ← 1
06:0030 | +020 0x7ffe9a0b6b00 ← 1
07:0038 | +028 0x7ffe9a0b6b08 → 0x7f86088643d90 (__libc_start_call_main+128) ← mov edi, eax
[ DISASM / x86-64 / set emulate on ]
0x556500ccf258 <read_file+8> sub rsp, 0x10 RSP ⇒ 0x7ffe9a0b6ad0 (0x7ffe9a0b6ae0 - 0x10)
0x556500ccf25c <read_file+12> mov esi, 0 ESI ⇒ 0
0x556500ccf261 <read_file+17> lea rax, [rip + 0xddb] RAX ⇒ 0x556500cd0043 ← 'flag.txt'
0x556500ccf268 <read_file+24> mov rdi, rax RDI ⇒ 0x556500cd0043 ← 'flag.txt'
0x556500ccf26b <read_file+27> mov eax, 0 EAX ⇒ 0
► 0x556500ccf270 <read_file+32> call open@plt <open@plt>
file: 0x556500cd0043 ← 'flag.txt'
oflag: 0
vararg: 0xfbad008b

0x556500ccf275 <read_file+37> mov dword ptr [rbp - 4], eax
0x556500ccf278 <read_file+40> cmp dword ptr [rbp - 4], -1
0x556500ccf27c <read_file+44> jne read_file+76 <read_file+76>

0x556500ccf27e <read_file+46> lea rax, [rip + 0xdc] RAX ⇒ 0x556500cd0050 ← '\nERROR: no flag found.\nPlease message the CAs.'
0x556500ccf285 <read_file+53> mov rdi, rax

```

- After opening the file, the program then moves on to read@plt, which reads the contents of flag.txt into a memory buffer. I carefully followed the execution and found the address where this buffer is located.

```

pwndbg> next
0x0000556500ccf27c in read_file ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
-----[ REGISTERS / show-flags off / show-compact-regs off ]-----
RAX 6
RBX 0
RCX 0x7f860872e53b (open64+91) ← cmp rax, -0x1000 /* 'H=' */
RDX 0
RDI 0xffffffffc
RSI 0x556500cd0043 ← 'flag.txt'
R8 0x7f8608835f10 (initial+16) ← 4
R9 0x7f860884f040 (__dl_fini) ← endbr64
R10 0
R11 0x246
R12 0x7ffe9a0b6c18 → 0x7ffe9a0b6e6b ← '/home/ctf/gdb2'
R13 0x556500ccf209 (main) ← endbr64
R14 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
R15 0x7f8608883040 (__rtld_global) → 0x7f86088842e0 → 0x556500cce000 ← 0x10102464c457f
RBP 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
RSP 0x7ffe9a0b6ad0 → 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
4
*RIP 0x556500ccf27c (read_file+44) ← jne 0x556500ccf29c
-----[ STACK ]-----
00:0000 | rsp 0x7ffe9a0b6ad0 → 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ←
endbr64
01:0008 | -008 0x7ffe9a0b6ad8 ← 0x600ccf30b
02:0010 | rbp 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
03:0018 | +008 0x7ffe9a0b6ae8 → 0x556500ccf230 (main+39) ← lea rax, [rip + 0xdd1]
04:0020 | +010 0x7ffe9a0b6af0 → 0x7ffe9a0b6c18 → 0x7ffe9a0b6e6b ← '/home/ctf/gdb2'
05:0028 | +018 0x7ffe9a0b6af8 ← 0x100000000
06:0030 | +020 0x7ffe9a0b6b00 ← 1
07:0038 | +028 0x7ffe9a0b6b08 → 0x7f8608643d90 (__libc_start_call_main+128) ← mov edi, eax
-----[ DISASM / x86-64 / set emulate on ]-----
0x556500ccf268 <read_file+24> mov rdi, rax RDI ⇒ 0x556500cd0043 ← 'flag.txt'
0x556500ccf26b <read_file+27> mov eax, 0 EAX ⇒ 0
0x556500ccf270 <read_file+32> call open@plt <open@plt>

0x556500ccf275 <read_file+37> mov dword ptr [rbp - 4], eax [0x7ffe9a0b6adc] ⇒ 6
0x556500ccf278 <read_file+40> cmp dword ptr [rbp - 4], -1 6 - -1 EFLAGS ⇒ 0x213 [ CF pf AF zf sf IF df of ]
► 0x556500ccf27c <read_file+44> ✓ jne read_file+76 <read_file+76>
↓
0x556500ccf29c <read_file+76> mov eax, dword ptr [rbp - 4] EAX, [0x7ffe9a0b6adc] ⇒ 6
0x556500ccf29f <read_file+79> mov edx, 0x80 EDX ⇒ 0x80
0x556500ccf2a4 <read_file+84> lea rcx, [rip + 0x2d95] RCX ⇒ 0x556500cd2040 (flag) ← 0
0x556500ccf2ab <read_file+91> mov rsi, rcx RSI ⇒ 0x556500cd2040 (flag) ← 0
0x556500ccf2ae <read_file+94> mov edi, eax EDI ⇒ 6

```

- To get the flag, I simply inspected the contents of this buffer after the read@plt call. The flag was sitting right there in memory.

```

pwndbg> next
0x0000556500ccf2b5 in read_file ()
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
-----[ REGISTERS / show-flags off / show-compact-regs off ]-----
+RAX 0x42
RBX 0
+RCX 0x7f860872e7e2 (read+18) ← cmp rax, -0x1000 /* 'H=' */
RDX 0x80
RDI 6
RSI 0x556500cd2040 (flag) ← 'flag{gl4d_y0u_f1gur3d_0ut_h0w_t0_f1nd_th3_fl4g!_e1305326dc862a3c}\n'
R8 0x7f8608835f10 (initial+16) ← 4
R9 0x7f860884f040 (_dl_fini) ← endbr64
R10 0
R11 0x246
R12 0x7ffe9a0b6c18 → 0x7ffe9a0b6e6b ← '/home/ctf/gdb2'
R13 0x556500ccf209 (main) ← endbr64
R14 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
R15 0x7f8608883040 (_rtld_global) → 0x7f86088842e0 → 0x556500cce000 ← 0x10102464c457f
RBP 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
RSP 0x7ffe9a0b6ad0 → 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
+RIP 0x556500ccf2b5 (read_file+101) ← lea rax, [rip + 0xdc3]
-----[ STACK ]-----
00:0000 | rsp 0x7ffe9a0b6ad0 → 0x556500cd1d90 (__do_global_dtors_aux_fini_array_entry) → 0x556500ccf1c0 (__do_global_dtors_aux) ← endbr64
01:0008 | -008 0x7ffe9a0b6ad8 ← 0x600ccf30b
02:0010 | rbp 0x7ffe9a0b6ae0 → 0x7ffe9a0b6b00 ← 1
03:0018 | +008 0x7ffe9a0b6ae8 → 0x556500ccf230 (main+39) ← lea rax, [rip + 0xdd1]
04:0020 | +010 0x7ffe9a0b6af0 → 0x7ffe9a0b6c18 → 0x7ffe9a0b6e6b ← '/home/ctf/gdb2'
05:0028 | +018 0x7ffe9a0b6af8 ← 0x100000000
06:0030 | +020 0x7ffe9a0b6b00 ← 1
07:0038 | +028 0x7ffe9a0b6b08 → 0x7f8608643d90 (__libc_start_call_main+128) ← mov edi, eax
-----[ DISASM / x86-64 / set emulate on ]-----
0x556500ccf29f <read_file+79> mov     edx, 0x80          EDX ⇒ 0x80
0x556500ccf2a4 <read_file+84> lea     rcx, [rip + 0x2d95] RCX ⇒ 0x556500cd2040 (flag) ← 0
0x556500ccf2ab <read_file+91> mov     rsi, rcx          RSI ⇒ 0x556500cd2040 (flag) ← 0
0x556500ccf2ae <read_file+94> mov     edi, eax          EDI ⇒ 6
0x556500ccf2b0 <read_file+96> call    read@plt          <read@plt>
► 0x556500ccf2b5 <read_file+101> lea     rax, [rip + 0xdc3]  RAX ⇒ 0x556500cd007f ← 0x443b031b0100
0x556500ccf2bc <read_file+108> mov     rdi, rax          RDI ⇒ 0x556500cd007f ← 0x443b031b0100
0x556500ccf2bf <read_file+111> call    puts@plt          <puts@plt>
0x556500ccf2c4 <read_file+116> nop
0x556500ccf2c5 <read_file+117> leave
0x556500ccf2c6 <read_file+118> ret

```

- I used “x/s 0x556500cd2040” to print the flag as it is a string.
- And like that, I had the flag:
flag{gl4d_y0u_f1gur3d_0ut_h0w_t0_f1nd_th3_fl4g!_e1305326dc862a3c}

```

pwndbg> x/s 0x556500cd2040
0x556500cd2040 <flag>: "flag{gl4d_y0u_f1gur3d_0ut_h0w_t0_f1nd_th3_fl4g!_e1305326dc862a3c}\n"
pwndbg>

```