Gdb

Code can be found here: https://github.com/c2003-tamu/413

Environment Setup

- Clone git repository
- Navigate to 413/gdb directory
- Ensure python is installed
- Ensure gdb is installed

Running the Script

- Run command: ./entry-point.sh
    - This will put you into gdb
    - Once in gdb, run command: findmain
        - This will display something like the following:

```
vboxuser@meow:~/Desktop/413/gdb$ ./entry-point.sh
Reading symbols from hw...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
(No debugging symbols found in hw)
(gdb) findmain
Function "_dl_start_user" not defined.
Breakpoint 1 (_dl_start_user) pending.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x00007ffff7fe4548 in _dl_start_user () from /lib64/ld-linux-x86-64.so.2
Breakpoint 2 at 0x7ffff7fe45a0

Breakpoint 2, 0x00007ffff7fe45a0 in _dl_start_user () from /lib64/ld-linux-x86-64.so.2
Breakpoint 3 at 0x55555555507f

Breakpoint 3, 0x000055555555507f in ?? ()
main found at address: 0x555555555151
Hello, World
[Inferior 1 (process 5075) exited normally]
(gdb) q
```

        - Compare this to gdb output, we can see that it is exactly the same:

```
vboxuser@meow:~/Desktop/413/gdb$ gdb unstrip
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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 unstrip...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
(No debugging symbols found in unstrip)
(gdb) b main
Breakpoint 1 at 0x1151
(gdb) r
Starting program: /home/vboxuser/Desktop/413/gdb/unstrip
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x0000555555555151 in main ()
(gdb)
```

- You should be able to use this script on any stripped c binary, just run the command: gdb -q -x findmain.py <stripped c binary>

Creating the Script

The script used for this assignment can be found in 413/gdb/findmain.py

To create this script, I followed the following workflow:

- I first used the tip presented in class where you put a breakpoint at _start
- Ended up tracing the return values to the _dl_start_user function, set a breakpoint there
- Analyzed the disassemble of _dl_start_user function, saw that _dl_start_user + 98 we make a jump

```
Dump of assembler code for function _dl_start_user:
=> 0x00007ffff7fe4548 <+0>:     mov    %rax,%r12
   0x00007ffff7fe454b <+3>:     mov    %rsp,%r13
   0x00007ffff7fe454e <+6>:     mov    0x19b14(%rip),%edx        # 0x7ffff7ffe068 <_rtld_global+4200>
   0x00007ffff7fe4554 <+12>:    test   $0x2,%edx
   0x00007ffff7fe455a <+18>:    je     0x7ffff7fe456d <_dl_start_user+37>
   0x00007ffff7fe455c <+20>:    mov    $0x1,%esi
   0x00007ffff7fe4561 <+25>:    mov    $0x5001,%edi
   0x00007ffff7fe4566 <+30>:    mov    $0x9e,%eax
   0x00007ffff7fe456b <+35>:    syscall
   0x00007ffff7fe456d <+37>:    mov    %edx,%edi
   0x00007ffff7fe456f <+39>:    and    $0xfffffffffffffff0,%rsp
   0x00007ffff7fe4573 <+43>:    call   0x7ffff7fdcb40 <_dl_cet_setup_features>
   0x00007ffff7fe4578 <+48>:    mov    %r12,%rax
   0x00007ffff7fe457b <+51>:    mov    %r13,%rsp
   0x00007ffff7fe457e <+54>:    mov    (%rsp),%rdx
   0x00007ffff7fe4582 <+58>:    mov    %rdx,%rsi
   0x00007ffff7fe4585 <+61>:    and    $0xfffffffffffffff0,%rsp
   0x00007ffff7fe4589 <+65>:    mov    0x18a70(%rip),%rdi        # 0x7ffff7ffd000 <_rtld_global>
   0x00007ffff7fe4590 <+72>:    lea    0x10(%r13,%rdx,8),%rcx
   0x00007ffff7fe4595 <+77>:    lea    0x8(%r13),%rdx
   0x00007ffff7fe4599 <+81>:    xor    %ebp,%ebp
   0x00007ffff7fe459b <+83>:    call   0x7ffff7fca780 <_dl_init>
   0x00007ffff7fe45a0 <+88>:    lea    -0x1a227(%rip),%rdx        # 0x7ffff7fca380 <_dl_fini>
   0x00007ffff7fe45a7 <+95>:    mov    %r13,%rsp
   0x00007ffff7fe45aa <+98>:    jmp    *%r12
   0x00007ffff7fe45ad <+101>:   nopl   (%rax)
End of assembler dump.
```

- decided to look at the registers before that point and saw that an address close to the main() function was in the r12 register

```
(gdb) info registers
rax            0x38                56
rbx            0x0                 0
rcx            0x7fffffffdec8      140737488346824
rdx            0x7fffffffdeb8      140737488346808
rsi            0x7ffff7ffe8b8      140737354131640
rdi            0x7ffff7ffe2e0      140737354130144
rbp            0x0                 0x0
rsp            0x7fffffffdeb0      0x7fffffffdeb0
r8             0x7ffffffe1f3       140737488347635
r9             0x7ffff7ffb440      140737354118208
r10            0x7fffffffdab0      140737488345776
r11            0x203               515
r12            0x555555555060      93824992235616
r13            0x7fffffffdeb0      140737488346800
r14            0x0                 0
r15            0x0                 0
rip            0x7ffff7fe45a0      0x7ffff7fe45a0 <_dl_start_user+88>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
fs_base        0x7ffff7fab740      140737353791296
gs_base        0x0                 0
```

- Set a breakpoint at the r12 value, continued to it
- Saw that we are calling an address extremely close to main at r12+31

```
(gdb) x/100i 0x555555555060
=> 0x555555555060:      endbr64
   0x555555555064:      xor    %ebp,%ebp
   0x555555555066:      mov    %rdx,%r9
   0x555555555069:      pop    %rsi
   0x55555555506a:      mov    %rsp,%rdx
   0x55555555506d:      and    $0xfffffffffffffff0,%rsp
   0x555555555071:      push   %rax
   0x555555555072:      push   %rsp
   0x555555555073:      xor    %r8d,%r8d
   0x555555555076:      xor    %ecx,%ecx
   0x555555555078:      lea    0xca(%rip),%rdi        # 0x555555555149
   0x55555555507f:      call   *0x2f53(%rip)        # 0x555555557fd8
   0x555555555085:      hlt
   0x555555555086:      cs nopw 0x0(%rax,%rax,1)
   0x555555555090:      lea    0x2f79(%rip),%rdi        # 0x555555558010
   0x555555555097:      lea    0x2f72(%rip),%rax        # 0x555555558010
   0x55555555509e:      cmp    %rdi,%rax
   0x5555555550a1:      je     0x5555555550b8
   0x5555555550a3:      mov    0x2f36(%rip),%rax        # 0x555555557fe0
   0x5555555550aa:      test   %rax,%rax
   0x5555555550ad:      je     0x5555555550b8
   0x5555555550af:      jmp    *%rax
   0x5555555550b1:      nopl   0x0(%rax)
   0x5555555550b8:      ret
   0x5555555550b9:      nopl   0x0(%rax)
   0x5555555550c0:      lea    0x2f49(%rip),%rdi        # 0x555555558010
   0x5555555550c7:      lea    0x2f42(%rip),%rsi        # 0x555555558010
   0x5555555550ce:      sub    %rdi,%rsi
```

- We can now just add 8 to this and we have our main address!

```
(gdb) x/10i 0x0000555555555151
=> 0x555555555151:      lea    0xeac(%rip),%rax        # 0x555555556004
   0x555555555158:      mov    %rax,%rdi
   0x55555555515b:      call   0x555555555050 <puts@plt>
   0x555555555160:      mov    $0x0,%eax
   0x555555555165:      pop    %rbp
   0x555555555166:      ret
   0x555555555167:      add    %dh,%bl
   0x555555555169:      nop    %edx
   0x55555555516c:      sub    $0x8,%rsp
   0x555555555170:      add    $0x8,%rsp
```

NOTE: In order to learn the python-gdb syntax, ChatGPT was used. Prompt: “Give me an overview on how to use python-gdb. Show me how to add breakpoints, see values in registers, and run other gdb commands such as r and c”. This was done in order to get a quick overview on how to use python-gdb without having to research for an inordinate amount of time.