### prerequisites

- Compile the program in gcc with debug symbols enabled (`-g`)
- Do NOT strip the binary
- To generate assembly code using gcc use the -S option: `gcc -S hello.c`

### utilities

#### objdump

1. Useful options

```
objdump --help
  -d, --disassemble        Display assembler contents of executable sections
  -S, --source             Intermix source code with disassembly
  -l, --line-numbers       Include line numbers and filenames in output
```

2. To analyze a binary, run:

```
objdump -d /path/to/binary
```

#### gdb

1. Disassemble

```
$ gdb -q ./a.out
Reading symbols from ./a.out...(no debugging symbols found)...done.
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x00000000004003a8  _init
0x00000000004003e0  __libc_start_main@plt
0x00000000004003f0  __gmon_start__@plt
0x0000000000400400  _start
0x0000000000400430  deregister_tm_clones
0x0000000000400460  register_tm_clones
0x00000000004004a0  __do_global_dtors_aux
0x00000000004004c0  frame_dummy
0x00000000004004f0  fce
0x00000000004004fb  main
0x0000000000400510  __libc_csu_init
0x0000000000400580  __libc_csu_fini
0x0000000000400584  _fini
(gdb) disassemble main
Dump of assembler code for function main:
   0x00000000004004fb <+0>:     push   %rbp
   0x00000000004004fc <+1>:     mov    %rsp,%rbp
   0x00000000004004ff <+4>:     sub    $0x10,%rsp
   0x0000000000400503 <+8>:     callq  0x4004f0 <fce>
   0x0000000000400508 <+13>:    mov    %eax,-0x4(%rbp)
   0x000000000040050b <+16>:    mov    -0x4(%rbp),%eax
   0x000000000040050e <+19>:    leaveq
   0x000000000040050f <+20>:    retq
End of assembler dump.
(gdb) disassemble /m main // update: /m is deprecated, use /s
Dump of assembler code for function main:
9       {
   0x00000000004004fb <+0>:     push   %rbp
   0x00000000004004fc <+1>:     mov    %rsp,%rbp
   0x00000000004004ff <+4>:     sub    $0x10,%rsp

10        int x = fce ();
   0x0000000000400503 <+8>:     callq  0x4004f0 <fce>
   0x0000000000400508 <+13>:    mov    %eax,-0x4(%rbp)

11        return x;
   0x000000000040050b <+16>:    mov    -0x4(%rbp),%eax

12      }
   0x000000000040050e <+19>:    leaveq
   0x000000000040050f <+20>:    retq

End of assembler dump.
(gdb)
```

The `/m` option in gdb is similar to option `-S` in objdump.

2. Disassemble by line range

```
(gdb) info line main
Line 3 of "main.c" starts at address 0x401050 <main> and ends at 0x401075 <main+
(gdb) disas 0x401050 0x401075
Dump of assembler code from 0x401050 to 0x401075:
0x00401050 <main+0>:    push   %ebp
0x00401051 <main+1>:    mov    %esp,%ebp
0x00401053 <main+3>:    sub    $0x18,%esp
0x00401056 <main+6>:    and    $0xfffffff0,%esp
0x00401059 <main+9>:    mov    $0x0,%eax
0x0040105e <main+14>:   add    $0xf,%eax
0x00401061 <main+17>:   add    $0xf,%eax
0x00401064 <main+20>:   shr    $0x4,%eax
0x00401067 <main+23>:   shl    $0x4,%eax
0x0040106a <main+26>:   mov    %eax,-0xc(%ebp)
0x0040106d <main+29>:   mov    -0xc(%ebp),%eax
0x00401070 <main+32>:   call   0x4010c4 <_alloca>
```

3. Examine as instructions

```
(gdb) x/i 0xdeadbeef
```

4. Show assembly instructions executed

```
(gdb) layout asm
    ┌────────────────────────────────────────────────────────────────────────────┐
    │0x7ffff740d756 <__libc_start_main+214>  mov    0x39670b(%rip),%rax        #
    │0x7ffff740d75d <__libc_start_main+221>  mov    0x8(%rsp),%rsi
    │0x7ffff740d762 <__libc_start_main+226>  mov    0x14(%rsp),%edi
    │0x7ffff740d766 <__libc_start_main+230>  mov    (%rax),%rdx
    │0x7ffff740d769 <__libc_start_main+233>  callq  *0x18(%rsp)
   >│0x7ffff740d76d <__libc_start_main+237>  mov    %eax,%edi
    │0x7ffff740d76f <__libc_start_main+239>  callq  0x7ffff7427970 <exit>
    │0x7ffff740d774 <__libc_start_main+244>  xor    %edx,%edx
    │0x7ffff740d776 <__libc_start_main+246>  jmpq   0x7ffff740d6b9 <__libc_start
    │0x7ffff740d77b <__libc_start_main+251>  mov    0x39ca2e(%rip),%rax        #
    │0x7ffff740d782 <__libc_start_main+258>  ror    $0x11,%rax
    │0x7ffff740d786 <__libc_start_main+262>  xor    %fs:0x30,%rax
    │0x7ffff740d78f <__libc_start_main+271>  callq  *%rax
```

~~~
multi-thre process 3718 In: __libc_start_main     Line: ??   PC: 0x7ffff740d76d
#3  0x00007ffff7466eb5 in _IO_do_write () from /lib/x86_64-linux-gnu/libc.so.6
#4  0x00007ffff74671ff in _IO_file_overflow ()
   from /lib/x86_64-linux-gnu/libc.so.6
#5  0x0000000000408756 in ?? ()
#6  0x0000000000403980 in ?? ()
#7  0x00007ffff740d76d in __libc_start_main ()
   from /lib/x86_64-linux-gnu/libc.so.6
(gdb)
```

5. When gdb stops, Display the disassembly of the next instruction (manually)

```
display/i $pc
x/i $pc

(gdb) help x
Examine memory: x/FMT ADDRESS.
ADDRESS is an expression for the memory address to examine.
FMT is a repeat count followed by a format letter and a size letter.
Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal),
  t(binary), f(float), a(address), i(instruction), c(char), s(string)
  and z(hex, zero padded on the left).
Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).
The specified number of objects of the specified size are printed
according to the format.

Defaults for format and size letters are those previously used.
Default count is 1.  Default address is following last thing printed
with this command or "print".
(gdb) x/10z 0x74c18b
0x74c18b:       0x6d    0x61    0x78    0x5f    0x63    0x68    0x61    0x72
0x74c193:       0x73    0x5f
(gdb) x/10c 0x74c18b
0x74c18b:       109 'm' 97 'a'  120 'x' 95 '_'  99 'c'  104 'h' 97 'a'  114 'r'
0x74c193:       115 's' 95 '_'
(gdb) x/xb 0x74c18b
0x74c18b:       0x6d
(gdb) x/20b 0x74c18b
0x74c18b:       0x6d    0x61    0x78    0x5f    0x63    0x68    0x61    0x72
0x74c193:       0x73    0x5f    0x66    0x69    0x6c    0x65    0x6e    0x61
0x74c19b:       0x6d    0x65    0x5f    0x78
```

6. Disassemble the entire next line when gdb stops (automatically)

```
set disassemble-next-line on
```

7. Show the backtrace stack

```
bt
info s
bt full
```

8. Show threads running and swithch to thread `n`

```
info threads
thread n
~~~

~~~~
```

9. Select frame `n`

```
frame n
```

10. Print data

```
info registers
info all-registers // useful for platform-specific regs
info registers r14
info locals
info args
info variables // all global and static variable names
info variables regex // global and static variables matching regex pattern
p $r14
p local_var_or_arg

(gdb) break main
(gdb) p &var // prints address of local var in main()
(gdb) break fun_1
(gdb) p &age // prints address of local var in fun_1()

(gdb) x/i $pc
=> 0x403cc0 <xstrlcpy+336>:     movdqa (%r14,%rcx,1),%xmm0
(gdb) p $xmm0
$7 = {v4_float = {0, 0, -nan(0x7fff00), 2.35098856e-38}, v2_double = {0, 7.2911220195561903e-304},
v16_int8 = {0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, -1, -1, -1, 0}, v8_int16 = {0, 0, 0, 0,
    -256, -1, -1, 255}, v4_int32 = {0, 0, -256, 16777215}, v2_int64 = {0, 72057594037927680}, uint128
= 0x00ffffffffffff000000000000000000}
```

#### readelf

1. grep a symbol

```
readelf -s a.out --wide |  grep token
```

2. Print the contents of rodata section as strings

```
readelf -p '.rodata' a.out
```

### resources

- [The Online Disassembler](https://onlinedisassembler.com)
- [godbolt](https://gcc.godbolt.org/)
- [How can I force GDB to disassemble?](https://stackoverflow.com/questions/1237489/how-can-i-force-
gdb-to-disassemble)
- [How to disassemble a binary executable in Linux to get the assembly code?]
(https://stackoverflow.com/questions/5125896/how-to-disassemble-a-binary-executable-in-linux-to-get-
the-assembly-code)
- [Show current assembly instruction in GDB](https://stackoverflow.com/questions/1902901/show-
current-assembly-instruction-in-gdb)
- [How to print register values in GDB?](https://stackoverflow.com/questions/5429137/how-to-print-
register-values-in-gdb)
- [Debugging with GDB](https://sourceware.org/gdb/onlinedocs/gdb/Machine-Code.html)
~~~~