UNM SCHOOL *of* ENGINEERING

Department of Computer Science

Lab 2: Examining Binary Programs

Prof. Patrick G. Bridges

Basic Binary Programs

- Programs have contents that are easier to understand using special tools
- Some simple tools
  - “hd” will print the binary contents of a program
  - “file” examines the header of a program to determine what type it is
  - “strings” prints the text strings in a file
- “objdump” is a powerful tool for generally inspecting binary objects (e.g. programs)
- “gdb” also has commands for inspecting and stepping through naked binaries

Objdump basics

- As we discussed in class, programs contain a sequence of “sections” that the OS loads
- The headers at the start of the object file describe these sections

```
> objdump -h a.out
a.out:     file format elf64-x86-64

Sections:
Idx Name     Size      VMA               LMA               File off  Algn
...
  5 .text     00091644  0000000000400360  0000000000400360 00000360  2**4
              CONTENTS, ALLOC, LOAD, READONLY, CODE
 23 .data    00001bd0  00000000006c0060  00000000006c0060  000c0060  2**5
              CONTENTS, ALLOC, LOAD, DATA
 24 .bss      00002518  00000000006c1c40  00000000006c1c40  000c1c30  2**5
              ALLOC
```

You can also look inside the sections themselves

- Disassemble text section:

```
> objdump –d a.out
000000000040105e <main>:
  40105e:     55                  push   %rbp
  40105f:     48 89 e5            mov    %rsp,%rbp
  401062:     bf 84 36 49 00      mov    $0x493684,%edi
  401067:     e8 c4 75 00 00      callq  408630 <_IO_puts>
  40106c:     5d                  pop    %rbp
  40106d:     c3                  retq
  40106e:     66 90               xchg   %ax,%ax
```

Can also look at the data itself

- Hex dump data (in this case the read only data)

```
>objdump –h
Idx Name    Size         VMA                      LMA                      File off     Algn
...
 9 .rodata   0001eae8  0000000000493680  0000000000493680  00093680  2**5
             CONTENTS, ALLOC, LOAD, READONLY, DATA
...
> objdump -s --start-address=0x493680 --stop-address=0x4936c0

a.out:     file format elf64-x86-64

Contents of section .rodata:
493680 01000200 48656c6c 6f20576f  726c6400  ....Hello World.
493690 6c696263 2d737461 72742e63 00464154  libc-start.c.FAT
4936a0 414c3a20 6b65726e 656c2074  6f6f206f  AL: kernel too o
4936b0 6c640a00 2f646576 2f757261  6e646f6d  ld../dev/urandom...
```

GDB with Assembly

- In addition to using GDB with C, it can also work with the underlying assembly code
- The object can often have function and variable names, even without source code

```
gdb> info functions
All defined functions:

File tmp.c:
int main(int, char **);

Non-debugging symbols:
0x080482f8  _init
0x08048340  printf@plt
...
```

So you can set breakpoints and watchpoint on these!

```
(gdb) break main
Breakpoint 1 at 0x4005a2
(gdb) watch i
Hardware watchpoint 2: i
(gdb) run
Starting program: /nfs/faculty/bridges/classes/CS341/a.out

Breakpoint 1, 0x00000000004005a2 in main ()
(gdb) c
Continuing.
Hardware watchpoint 2: i

Old value = 0
New value = 1
0x00000000004005d5 in main ()
(gdb)
```

And you can look at what the hardware is doing

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x000000000040059e <+0>:       push   %rbp
   0x000000000040059f <+1>:       mov    %rsp,%rbp
   0x00000000004005a2 <+4>:       sub    $0x10,%rsp
   …
(gdb) info registers
rax        0x1              1
rbx        0x0              0
rcx        0x1              1
rdx        0x7ffff7dd59e0            140737351866848
rsi        0x7ffffffe       2147483646
rdi        0x7ffff7ff5005            140737354092549
rbp        0x7fffffffe790            0x7fffffffe790
rsp        0x7fffffffe780            0x7fffffffe780
...
```

Including single stepping in assembly

```
(gdb) display/i $pc
1: x/i $pc
=> 0x4005d5 <main+55>:    mov    0x200a71(%rip),%eax        # 0x60104c <i>
(gdb) stepi
0x00000000004005db in main ()
1: x/i $pc
⇒ 0x4005db <main+61>:     cmp    $0x9,%eax
(gdb) p $eax
$1 = 1
```

Sometimes you have to coerce the type system

- From GDB’s perspective, what’s in memory or registers are just numbers; may need to cast

```
(gdb) disassemble do_something
Dump of assembler code for function do_something:
   0x000000000040057d <+0>:      push   %rbp
  …
   0x0000000000400592 <+21>:     mov    $0x0,%eax
(gdb) break 0x400592
Function "0x400592" not defined.
Make breakpoint pending on future shared library load? (y or [n]) n
(gdb) break *(void*)0x400592
Breakpoint 4 at 0x400592
(gdb) p $edi
$6 = 4195956
(gdb) p (char *)$edi
$7 = 0x400674 "%d... "
```