COSC 458-647
Application Software Security

GDB Debugger in Linux

What is Gdb?

gdb is the GNU Project debugger

- gdb provides some helpful functionality
  - Allows you to stop your program at any given point.
  - You can examine the state of your program when it’s stopped.
  - Change things in your program, so you can experiment with correcting the effects of a bug.
- Also a command-line program
- Is also available on spinlock/coredump

Using gdb

- To start gdb with your hello program type:

  gdb <your_program_name>

- When gdb starts, your program is not actually running.

- You have to use the ***run*** command to start execution.

- Before you do that, you should place some break points.

- Once you hit a break point, you can examine any variable.

Compile your C program

- Let's do it: `hello.c`

```
#include <stdio.h>
int main() {
        int i;
        for (i=0; i <=10; i++) {
                printf("Hello World");
        }
}
```

- Compile
  - **`gcc -g -o hello_run hello.c`**

objdump

- **objdump** displays information about one or more object files.
  - The options control what particular information to display.
  - This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work.

- -d
  - --disassemble
  - Display the assembler mnemonics for the machine instructions from objfile. This option only disassembles those sections which are expected to contain instructions.
- -D
  - --disassemble-all
  - Like -d, but disassemble the contents of all sections, not just those expected to contain instructions.
  - --prefix-addresses
  - When disassembling, print the complete address on each line. This is the older disassembly format.

objdump
[**-a**|**--archive-headers**]
[**-b** *bfdname*|**--target=***bfdname*]
[**-C**|**--demangle**[*=style*]]
[**-d**|**--disassemble**]
[**-D**|**--disassemble-all**]
[**-z**|**--disassemble-zeroes**]
[**-EB**|**-EL**|**--endian=**{big | little }]
[**-f**|**--file-headers**]
[**--file-start-context**]
[**-g**|**--debugging**]
[**-h**|**--section-headers**|**--headers**]
[**-i**|**--info**]
[**-j** *section*|**--section=***section*]
[**-l**|**--line-numbers**]
[**-S**|**--source**]
[**-m** *machine*|**--architecture=***machine*]
[**-M** *options*|**--disassembler-options=***options*]
[**-p**|**--private-headers**]
[**-r**|**--reloc**]
[**-R**|**--dynamic-reloc**]
[**-s**|**--full-contents**]
[**-G**|**--stabs**]
[**-t**|**--syms**]
[**-T**|**--dynamic-syms**]
[**-x**|**--all-headers**]
[**-w**|**--wide**]
[**--start-address=***address*]
[**--stop-address=***address*]
[**--prefix-addresses**]
[**--[no-]show-raw-insn**]
[**--adjust-vma=***offset*]
[**-V**|**--version**]
[**-H**|**--help**]

objfile...

objdump

- **objdump** displays information about one or more object files.

- -M options
  - --disassembler-options=options
  - Pass target specific information to the disassembler. Only supported on some targets.

objdump
[**-a**|**--archive-headers**]
[**-b** *bfdname*|**--target=***bfdname*]
[**-C**|**--demangle**[*=style*]]
[**-d**|**--disassemble**]
[**-D**|**--disassemble-all**]
[**-z**|**--disassemble-zeroes**]
[**-EB**|**-EL**|**--endian=**{big | little }]
[**-f**|**--file-headers**]
[**--file-start-context**]
[**-g**|**--debugging**]
[**-h**|**--section-headers**|**--headers**]
[**-i**|**--info**]
[**-j** *section*|**--section=***section*]
[**-l**|**--line-numbers**]
[**-S**|**--source**]
[**-m** *machine*|**--architecture=***machine*]
[**-M** *options*|**--disassembler-options=***options*]
[**-p**|**--private-headers**]
[**-r**|**--reloc**]
[**-R**|**--dynamic-reloc**]
[**-s**|**--full-contents**]
[**-G**|**--stabs**]
[**-t**|**--syms**]
[**-T**|**--dynamic-syms**]
[**-x**|**--all-headers**]
[**-w**|**--wide**]
[**--start-address=***address*]
[**--stop-address=***address*]
[**--prefix-addresses**]
[**--[no-]show-raw-insn**]
[**--adjust-vma=***offset*]
[**-V**|**--version**]
[**-H**|**--help**]

objfile...

objdump

- objdump –M intel –D hello_run | grep –A20 main.:
  - -M intel: Set disassembly options to Intel assembly

Intex Syntax		AT&T Syntax	
`mov`	`eax,1`	`movl`	`$1,%eax`
`mov`	`ebx,0ffh`	`movl`	`$0xff,%ebx`
`int`	`80h`	`int`	`$0x80`

  - -D hello_run: Disassemble our hello_run program
  - grep –A20 main.: Get 20 instructions from main() function

Set Intel assembler as default

- Write it to file gdbinit

```
echo “set disassembly intel” > ~/.gdbinit
```

- Check

```
cat ~/.gdbinit
```

Useful gdb commands

- **gdb -q** <program_name>
  - -q (quiet option) tells GDB not to print version information on startup.

- (gdb) **list**
  - List the actual code of the program

- (gdb) **disassemble** <func_name>
  - Dump of assembler code for function <func_name>

- (gdb) **quit**

Useful gdb commands

- **run** *<command-line-arguments>*
  - Begin execution of your program with arguments

- **break** *<place>*
  - *place* can be the name of a function or a line number
  - For example: **break main** will stop execution at the first instruction of your program

- **delete** *<N>*
  - Removes breakpoints, where *N* is the number of the breakpoint

- **step**
  - Executes current instruction and stops on the next one

gdb commands cont.

- **`next`**
  - Same as **step** except this doesn't step into functions
- **`nexti`**
  - Execute the next instruction
- **`Print`** *`E`*
  - Prints the value of any variable in your program when you are at a breakpoint, where *E* is the name of the variable you want to print
- **`help`** *`command`*
  - Gives you more information about any command or all if you leave out command

Program Execution commands

- [b]reak <function name or filename:line# or *memory address>
  - Sets a breakpoint on either a function, a line given by a line number, or the instruction located at a particular address.
  - If you do not have access to the source code of a function and wish to set a breakpoint on a particular instruction, call disassemble function name (where function name is the nameof the procedure); this command will allow you to see the memory address of each instruction.
- Example

```
(gdb) break main
Breakpoint 1 at 0x80488f6: file main.c, line 67.
```

Program Execution commands

- [d]elete <breakpoint #>
  - Removes the indicated breakpoint. To see breakpoint numbers, run info break, or i b.

```
(gdb) delete 4
```

- [condition] <breakpoint #> <condition>
  - Updates the breakpoint indicated by the given number so that execution of the program stops at that point only if condition is true. condition is expressed in C syntax, and can only use variables and functions that are available in the scope of the breakpoint location.

```
(gdb) break main
Breakpoint 1 at 0x80488f6: file main.c, line 48
(gdb) condition 1 argc <= 2 || !strcmp(argv[1], "jasmine")
```

Program Execution commands

- `[i]nfo (about)`

Lists information about the argument `(about)`, or lists what possible arguments are if none is provided.

`(about)` can be one of the following:

- `[f]rame` - list information about the current stack frame, including the address of the current frame, the address of the previous frame, locations of saved registers, function arguments, and local variables.
- `[s]tack` - list the stack backtrace, showing what function calls have been made, and their arguments. You can also use the commands backtrace or where to do the same.
- `[r]egisters` - lists the contents of each register. [all-r]egisters lists even more registers.
- `[b]reak` - lists the number and address of each breakpoint, and what function the breakpoint is in.
- `[fu]nctions` - lists all of the function signatures, if the program was compiled with the gcc flag -g. This is useful for setting breakpoints in functions.

Program Execution commands

- `[i]nfo (about)`

Lists information about the argument `(about)`, or lists what possible arguments are if none is provided.

```
info registers
info register eip
info register esp
```

Program Execution commands

- `[r]un (arg1 arg2 ... argn)`
  - Runs the loaded executable program with program arguments arg1 ... argn.
- `[c]ontinue`
  - Resumes execution of a stopped program, stopping again at the next breakpoint.
- `[s]tep`
  - Steps through a single line of code. Steps into function calls.

```
(gdb) break main
Breakpoint 1 at 0x8049377: file main.c, line 34.
(gdb) r
Breakpoint 1, main (argc=2, argv=0xbffff704) at main.c:34
35 int val = foo(argv[1]);
(gdb) s
foo (word=0xbffff8b3) at main.c:11
12 char bar = word[0];
```

Program Execution commands

- `[n]ext`
  - Steps through a single line of code. Steps over function calls.

- `[n]ext[i]`
  - Steps through a single x86 instruction. Steps over calls.

Viewing Variables, Registers and Memory

- [p]rint <expression>
  - Prints the value which the indicated expression evaluates to. expression can contain variable names (from the current scope), memory addresses, registers, and constants as its operands to various operators. It is written in C syntax, which means that in addition to arithmetic operations, you can also use casting operations and dereferencing operations.

```
(gdb) print *(char *)($esp + $eax + my_ptr_array[13])
'e'
```

- [p]rint/x <expression>
  - Prints the value which the indicated expression evaluates to as a hexadecimal number. Expression is evaluated the same way as it is in print.

```
(gdb) p/x my_var
$1 = 0x1b
```

Examinning memory/register/variable

- `[x]/(number)(format)(unit size) <address>`
  - Examines the data located in memory at address.
  - **`number`** optionally indicates that several contiguous elements, beginning at address, should be examined. This is very useful for examining the contents of an array. By default, this argument is 1.
  - **`format`** indicates how data should be printed. In most cases, this is the same character that you would use in a call to `printf()`. One exception is the format `i`, which prints an instruction rather than a decimal integer.
  - **`unit size`** indicates the size of the data to examine. It can be
    - [b]ytes,
    - [h]alfwords (2 bytes),
    - [w]ords, or
    - [g]iant words.
    - By default, this is bytes, which is perfect for examining instructions.

Backtrace

- [b]ack[t]race
  - Prints a stack trace, listing each function and its arguments. This does the same thing as the commands info stack and where.

```
(gdb) bt
#0 fibonacci (n=1) at main.c:45
#1 fibonacci (n=2) at main.c:45
#2 fibonacci (n=3) at main.c:45
#3 main (argc=2, argv=0xbffff6e4) at main.c:34
```

- [where]
  - Prints a stack trace, listing each function and its arguments. This is the same as the commands info stack and backtrace.