GDB Primer

Thursday, February 12th, 2015

Stack discipline: home exercise

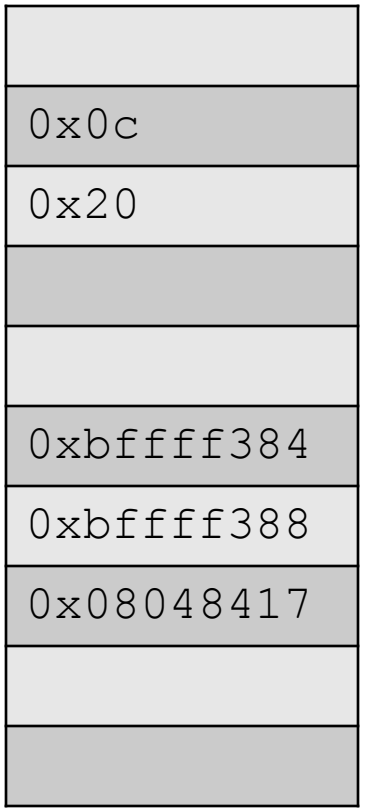
```
swap:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        movl    (%eax), %eax
        movl    %eax, -4(%ebp)
        movl    12(%ebp), %eax
        movl    (%eax), %edx
        movl    8(%ebp), %eax
        movl    %edx, (%eax)
        movl    12(%ebp), %eax
        movl    -4(%ebp), %edx
        movl    %edx, (%eax)
        leave
        ret
```

Register File

%ebp	0xbffff38c
%esp	0xbffff370
%eax	
%edx	
…	…

Task:
Now, walking through each line, modify the stack and register file accordingly….

Stack



GDB

- Step through program execution
- Examine values of program variables
- Set breakpoints to halt execution at any point
- Watch variables to see when they change

Generating Assembly

- Getting the assembly code:

```
% gcc -m32 -S simple.c
```

- Resulting file:

```
simple.s
```

Using GDB

- Compile for debugging:

```
% gcc -g main.c -o main
```

Note: code depends on system

- To debug:

```
% gdb main
```

GDB: Example

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5     int a, b, c;
6     a = 4;
7     b = 10;
8     c = a*b;
9     char* d = "My string";
10
11 printf("A is %d, B is %d,
          C is %d\nD is %s\n",
          a, b, c, d);
12
13 return 0;
14 }
```

```
(gdb) break simple.c:8
Breakpoint 1 at 0x80483dd: file simple.c,line 8.
(gdb) run
Starting program: /home/…/…/a.out
Breakpoint 1, main () at simple.c:8
8 c = a*b;
(gdb) print a
$1 = 4
(gdb) print b
$2 = 10
(gdb) print c
$3 = 134513642
(gdb) where
#0 main () at simple.c:8
(gdb) continue
Continuing.
A is 4, b is 10, and c is 40
Program exited normally.
```

Select Commands

- help
- break
- info breakpoints
- next / nexti
- step / stepi
- finish
- x/c ; x/d ; x/s
- print

- Run
- continue
- Info registers
- disassemble / disass
- kill
- quit / q

- See: gdb commands

Special Registers

- `%esp` : Stack Pointer
- `%ebp` : Stack Base/Frame Pointer
- `%eax` : Holds function return value
- `%eip` : Holds address of next instruction

Bomblab

- Inspect pointers, registers, stack directly
  - bomb* already compiled for debugging
- See moodle for short command reference
- Now, let's defuse phase_1

Questions?