CS 449 Rec 6

Slides are available at https://github.com/wenyuli08/CS449_Rec_Fall2020

Midterm due today @ 6:00 pm

Schedule

- Scope & lifetime
- gdb
- Bomb lab

Scope: “Where a name can be seen”

Global scope	Can be seen everywhere in any file
File scope	Can be seen only in current file
Function prototype scope (for C)	Declared in prototype, so only visible in prototype Int addition(int a, int b);
Function scope	The “local”
Block scope	{} indicates a block in C

Lifetime: “The time between allocation and deallocation”

In other words, it’s how long a variable’s memory is valid

For simplicity, we divide it into Global and Local

```
int glob = 0;
int main() {
    int m = 10;
    func();
    return 0;
}
```

```
void func() {
    int f = 10;
}
```

m

glob

f

(Program start)	Main running	Func running	Main running	(Program ends)

time

```
char* func() {
    char str[50] =
"hello world";
    return str;
}

int main() {
    char* str = func();
    printf("%s\n", str);
    return 0;
}
```

Be careful: pointer variables and whatever they point to have **separate lifetime**

- What's the lifetimes of these variables?
  - Char *str
  - Char str
- What does this print?

Conclusion/Difference (briefly describe)

Examples that a variable is within scope but not lifetime

Examples that a variable is within lifetime but not scope

Quiz (1/3)

- What is it doing?
- How many args?
- What (types) are they?

```
pushq   %rbp
movq    %rsp, %rbp
movq    %rdi, -24(%rbp)
movl    %esi, -28(%rbp)
...
popq    %rbp
ret
```

(2/3)

- What is it doing?
- (L2 is whatever happens right after it)

```
        movl    $0, -8(%rbp)
.L3:
        movl    -8(%rbp), %eax
        cmpl    -28(%rbp), %eax
        jge     .L2
        addl    $1, -4(%rbp)
        addl    $1, -8(%rbp)
        jmp     .L3
.L2:
```

(3/3)

- What is it doing now that you have the whole picture?

```
foo(int*, int):
        pushq   %rbp
        movq    %rsp, %rbp
        movq    %rdi, -24(%rbp)
        movl    %esi, -28(%rbp)
        movq    -24(%rbp), %rax
        movl    (%rax), %eax
        movl    %eax, -4(%rbp)
        movl    $0, -8(%rbp)
.L3:
        movl    -8(%rbp), %eax
        cmpl    -28(%rbp), %eax
        jge     .L2
        addl    $1, -4(%rbp)
        addl    $1, -8(%rbp)
        jmp     .L3
.L2:
        movl    -4(%rbp), %eax
        popq    %rbp
        ret
```

```c
int foo(int *p, int val) {
    int ret = *p;
    int i;
    for (i = 0; i < val; i ++) {
        ret ++;
    }
    return ret;
}
```

What could this be?

```
.L3:
        movl    -4(%rbp), %eax
        cmpl    -28(%rbp), %eax
        jge     .L2/jg…
        (…)
        jmp     .L3
```

The Intel Syntax (bomb lab)

```
foo(int*, int):
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-24], rdi
        mov     DWORD PTR [rbp-28], esi
        mov     rax, QWORD PTR [rbp-24]
        mov     eax, DWORD PTR [rax]
        mov     DWORD PTR [rbp-4], eax
        mov     DWORD PTR [rbp-8], 0
.L3:
        mov     eax, DWORD PTR [rbp-8]
        cmp     eax, DWORD PTR [rbp-28]
        jge     .L2
        add     DWORD PTR [rbp-4], 1
        add     DWORD PTR [rbp-8], 1
        jmp     .L3
.L2:
        mov     eax, DWORD PTR [rbp-4]
        pop     rbp
        ret
```

```
foo:
  movl    $0,      %eax

L1:
  testq   %rdi,    %rdi
  je      L2
  movq    (%rdi),  %rdi
  addl    $1,       %eax
  jmp     L1

L2:
  ret
```

```
int foo(long* p) {
  int result =_____;
  while (_________) {
    p =____________;
    ______=___________;
  }
  return result;
}
```

Assembly Instructions

`mov a, b`	Copy from `a` to `b`.
`movs a, b`	Copy from `a` to `b` with sign extension. Needs *two* width specifiers.
`movz a, b`	Copy from `a` to `b` with zero extension. Needs *two* width specifiers.
`lea a, b`	Compute address and store in `b`. *Note:* the scaling parameter of memory operands can only be 1, 2, 4, or 8.
`push src`	Push `src` onto the stack and decrement stack pointer.
`pop dst`	Pop from the stack into `dst` and increment stack pointer.
`call <func>`	Push return address onto stack and jump to a procedure.
`ret`	Pop return address and jump there.
`add a, b`	Add from `a` to `b` and store in `b` (and sets flags).
`sub a, b`	Subtract `a` from `b` (compute `b-a`) and store in `b` (and sets flags).
`imul a, b`	Multiply `a` and `b` and store in `b` (and sets flags).
`and a, b`	Bitwise AND of `a` and `b`, store in `b` (and sets flags).
`sar a, b`	Shift value of `b` *right* (*arithmetic*) by `a` bits, store in `b` (and sets flags).
`shr a, b`	Shift value of `b` *right* (*logical*) by `a` bits, store in `b` (and sets flags).
`shl a, b`	Shift value of `b` *left* by `a` bits, store in `b` (and sets flags).
`cmp a, b`	Compare `b` with `a` (compute `b-a` and set condition codes based on result).
`test a, b`	Bitwise AND of `a` and `b` and set condition codes based on result.
`jmp <label>`	Unconditional jump to address.
`j* <label>`	Conditional jump based on condition codes (*more on next page*).
`set* a`	Set byte based on condition codes.

Part 2: Follow the execution of foo in assembly, where `0x1000` is passed in to `%rdi`

Write the values of `%rdi` and `%eax` in the columns. If the value doesn't change, you can leave it blank

Instruction	%rdi (hex)	%eax (decimal)
movl	0x1000	0
testq		
je		

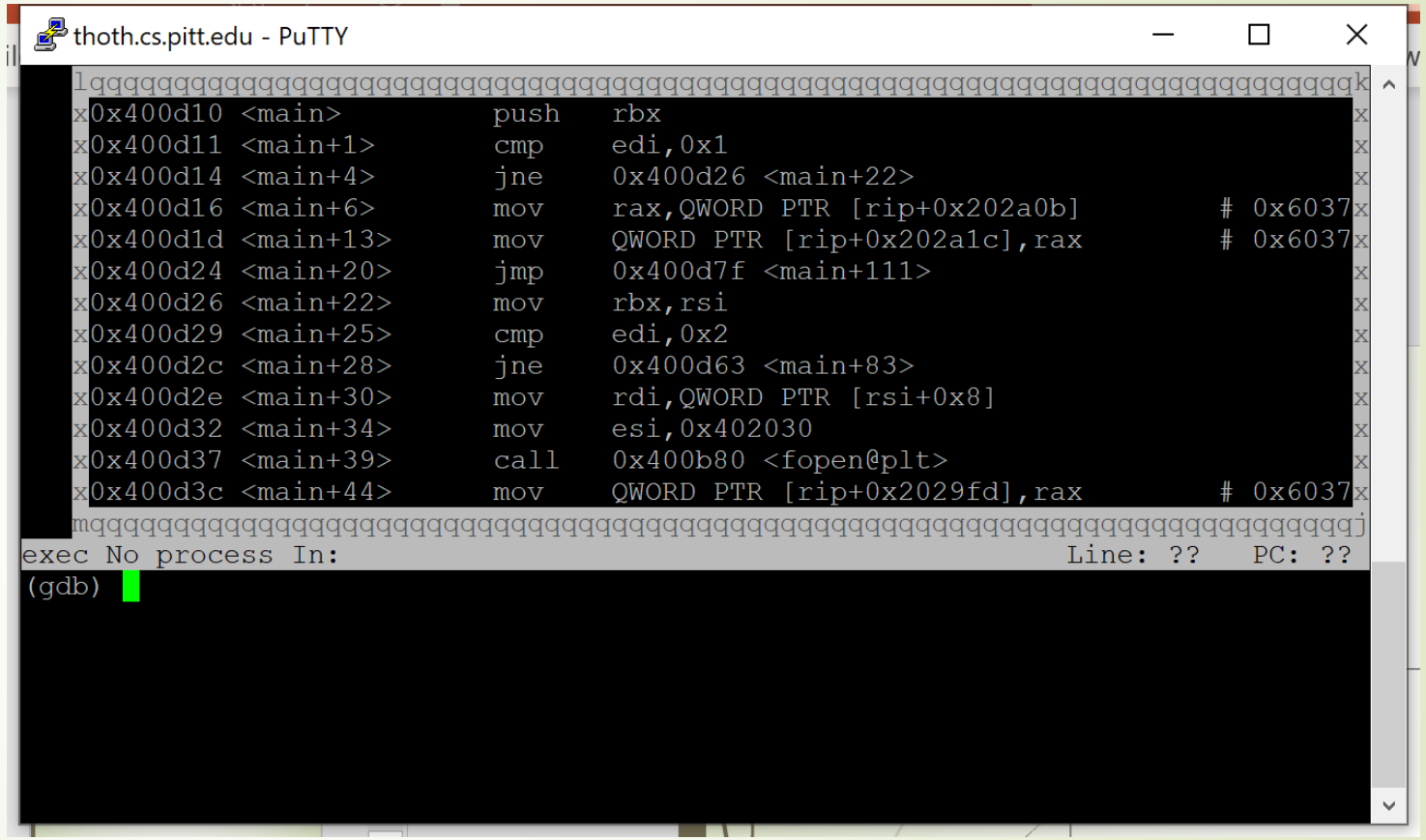
Address	Value
0x1000	0x1030
0x1008	0x1020
0x1010	0x1000
0x1018	0x0000
0x1020	0x1030
0x1028	0x1008
0x1030	0x0000
0x1038	0x1038
0x1040	0x1048
0x1048	0x1040

```
int foo(long* p) {
  int result = 0;
  while (p != NULL) {
    p = *(long**)p;
    result = result + 1;
  }
  return result;
}
```

Instruction	%rdi (hex)	%eax (decimal)
movl	0x1000	0
testq		
je		
movq	0x1030	
addl		1
jmp		
testq		
je		
movq	0x0	
addl		2
jmp		
testq		
je		
ret		

GDB (basics in rec 1)

- layout next lets you see the code as you step through <u>(at the beginning)</u>
- Don't do this during defusing
- Close it with ctrl+X+A



Read the pdf:
Hints, where to start?

Useful break points to set

- **Break explode_bomb (for all phases)**
- …not_equal/…compare/…
- Break … if *cond*
- *Break printf("bad news")*

thoth.cs.pitt.edu - PuTTY

```
(27) thoth $ gdb bomb
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-64.el6_5.2)
Copyright (C) 2010 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /afs/pitt.edu/home/k/m/kmc51/private/bomb2/bomb...done.
(gdb) break explode_bomb
Breakpoint 1 at 0x401248
(gdb) run
Starting program: /afs/pitt.edu/home/k/m/kmc51/private/bomb2/bomb
Welcome to my fiendish little bomb. You have 4 phases with
which to blow yourself up. Have a nice day!
test string which probably won't work

Breakpoint 1, 0x0000000000401248 in explode_bomb ()
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.212.el6_10.3.x86_64
(gdb) quit
A debugging session is active.

        Inferior 1 [process 423] will be killed.

Quit anyway? (y or n) y
(28) thoth $
```

Useful strings to search for (for simple bomb)

- “Phase 1 defused. Cool. How about the next one?” / “Awesome! That's number 2. Keep going!” / …
- Especially near compare/if statement

Disas = disassembly

thoth.cs.pitt.edu - PuTTY

```
[thoth ~/private/cs449_ta/lab4/bomb66]: gdb ./bomb
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-64.el6_5.2)
Copyright (C) 2010 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /afs/pitt.edu/home/w/e/wel104/private/cs449_ta/lab4/bomb66/
bomb...done.
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e10 <+0>:     sub    rsp,0x8
   0x0000000000400e14 <+4>:     mov    esi,0x4021c8
   0x0000000000400e19 <+9>:     call   0x400fe6 <strings_not_equal>
   0x0000000000400e1e <+14>:    test   eax,eax
   0x0000000000400e20 <+16>:    je     0x400e27 <phase_1+23>
   0x0000000000400e22 <+18>:    call   0x401238 <explode_bomb>
   0x0000000000400e27 <+23>:    add    rsp,0x8
   0x0000000000400e2b <+27>:    ret
End of assembler dump.
(gdb)
```

Objdump, strings, apropos, info, man… (called directly)

thoth.cs.pitt.edu - PuTTY

```
[thoth ~/private/cs449_ta/lab4/bomb66]: objdump -t bomb

bomb:     file format elf64-x86-64

SYMBOL TABLE:
0000000000400200 l    d  .interp        0000000000000000              .interp
000000000040021c l    d  .note.ABI-tag  0000000000000000              .note.ABI-
tag
000000000040023c l    d  .note.gnu.build-id     0000000000000000              .n
ote.gnu.build-id
0000000000400260 l    d  .gnu.hash      0000000000000000              .gnu.hash
0000000000400290 l    d  .dynsym        0000000000000000              .dynsym
0000000000400590 l    d  .dynstr        0000000000000000              .dynstr
00000000004006c0 l    d  .gnu.version   0000000000000000              .gnu.versi
on
0000000000400700 l    d  .gnu.version_r 0000000000000000              .gnu.versi
on_r
0000000000400740 l    d  .rela.dyn      0000000000000000              .rela.dyn
00000000004007a0 l    d  .rela.plt      0000000000000000              .rela.plt
0000000000400a28 l    d  .init  0000000000000000              .init
0000000000400a40 l    d  .plt   0000000000000000              .plt
0000000000400c00 l    d  .text  0000000000000000              .text
0000000000401ffc l    d  .fini  0000000000000000              .fini
0000000000402020 l    d  .rodata        0000000000000000              .rodata
```

thoth.cs.pitt.edu - PuTTY

```
(42) thoth $ objdump -d bomb > bomb_objdump_d.txt
(43) thoth $ head bomb_objdump_d.txt

bomb:     file format elf64-x86-64


Disassembly of section .init:

0000000000400a28 <_init>:
  400a28:       48 83 ec 08             sub    $0x8,%rsp
  400a2c:       e8 fb 01 00 00          callq  400c2c <call_gmon_start>
  400a31:       48 83 c4 08             add    $0x8,%rsp
(44) thoth $ objdump -t bomb | grep -n -e explode
98:0000000000401248 g     F .text       0000000000000036              explode_bomb
(45) thoth $
```

Stepping through and displaying next step

thoth.cs.pitt.edu - PuTTY

```
which to blow yourself up. Have a nice day!
test

Breakpoint 1, 0x0000000000400fe6 in strings_not_equal ()
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.212.el6_10.3.x8
6_64
(gdb) display /i $rip
1: x/i $rip
=> 0x400fe6 <strings_not_equal>:        push   r12
(gdb) display /i $rip
2: x/i $rip
=> 0x400fe6 <strings_not_equal>:        push   r12
(gdb) step
Single stepping until exit from function strings_not_equal,
which has no line number information.
0x0000000000400e1e in phase_1 ()
2: x/i $rip
=> 0x400e1e <phase_1+14>:       test   eax,eax
1: x/i $rip
=> 0x400e1e <phase_1+14>:       test   eax,eax
(gdb) display /i $rip
3: x/i $rip
=> 0x400e1e <phase_1+14>:       test   eax,eax
(gdb)
```

Watch point pause the program whenever the value changes

PuTTY (inactive)

```
0000000000401033 g     F .text  00000000000000c1              initialize_bomb
0000000000000000       F *UND*  0000000000000000              __ctype_b_loc@@GLI
BC_2.3
0000000000603730 g     O .bss   0000000000000008              stderr@@GLIBC_2.2.
5
0000000000000000       F *UND*  0000000000000000              socket@@GLIBC_2.2.
5


[thoth ~/private/cs449_ta/lab4/bomb66]: gdb ./bomb
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-64.el6_5.2)
Copyright (C) 2010 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /afs/pitt.edu/home/w/e/wel104/private/cs449_ta/lab4/bomb66/
bomb...done.
(gdb) watch string_length
Hardware watchpoint 1: string_length
(gdb)
```

Recall that, prior to the call to `strings_not_equal`, the arguments for `strings_not_equal` are pushed onto the registers `rsi` and `rdi` (in particular, these are the registers used to place the first and second function arguments, respectively). Inspect the contents of those registers to learn something about them:

```
x /2wx $rdi
```

Alternatively, you may find it more conveniently to inspect the register contents using a string format.

```
x /s $rdi
```

This should reveal something. How about the other register? What is it used for? Now run the same commands to inspect the register `$rsi`.

- Or print
- Or x/s

8. Once you've stopped at a breakpoint, say at a `cmp` machine instruction, step through machine instructions using `stepi (si)` or `nexti (ni)` and try to understand what happens (what each number is compared against, what conditions are being checked)

```
(gdb) break strings_not_equal
Breakpoint 1 at 0x400ff6
(gdb) run
Starting program: /afs/pitt.edu/home/k/m/kmc51/private/bomb2/bomb
Welcome to my fiendish little bomb. You have 4 phases with
which to blow yourself up. Have a nice day!
test

Breakpoint 1, 0x0000000000400ff6 in strings_not_equal ()
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.212.el6_10.3.x86_64
(gdb) si
0x0000000000400ff8 in strings_not_equal ()
(gdb) si
0x0000000000400ff9 in strings_not_equal ()
(gdb) display /i $rip
1: x/i $rip
=> 0x400ff9 <strings_not_equal+3>:      push   %rbx
(gdb) si
0x0000000000400ffa in strings_not_equal ()
1: x/i $rip
=> 0x400ffa <strings_not_equal+4>:      mov    %rdi,%rbx
(gdb)
```

Notes

Guess it like you were to implement it

Draw the function call flow

1/0 values may indicate comparison

Keep a paper/Word doc to keep track

Sometimes things are implicit, but it shouldn't matter

Good luck!

It’s possible (not easy) to get all answers w/o exploding the bomb