Assignment 2

Tuesday, September 24, 2024 8:59 PM

From text book chapter 3.1 - 3.12 except 3.11

Phase - 1

0	1	2	3	4	5	6	7
M	#	?	Y	J			

```
Dump of assembler code for function bombphase_1:
=> 0x000000000040151a <+0>:     sub    $0x8,%rsp
   0x000000000040151e <+4>:     cmpb   $0x4d,(%rdi)
   0x0000000000401521 <+7>:     jne    0x401564 <bombphase_1+74>
   0x0000000000401523 <+9>:     cmpb   $0x3f,0x2(%rdi)
   0x0000000000401527 <+13>:    jne    0x401564 <bombphase_1+74>
   0x0000000000401529 <+15>:    cmpb   $0x6a,0x4(%rdi)
   0x000000000040152d <+19>:    jne    0x401564 <bombphase_1+74>
   0x000000000040152f <+21>:    cmpb   $0x79,0x3(%rdi)
   0x0000000000401533 <+25>:    jne    0x401564 <bombphase_1+74>
   0x0000000000401535 <+27>:    cmpb   $0x23,0x1(%rdi)
   0x0000000000401539 <+31>:    je     0x401572 <bombphase_1+88>
   0x000000000040153b <+33>:    cmp    %sil,0xf(%rdi)
   0x000000000040153f <+37>:    jne    0x401564 <bombphase_1+74>
   0x0000000000401541 <+39>:    cmp    %cl,0x17(%rdi)
   0x0000000000401544 <+42>:    jne    0x401564 <bombphase_1+74>
   0x0000000000401546 <+44>:    movzbl 0x14(%rdi),%edx
   0x000000000040154a <+48>:    movzbl 0x24(%rdi),%eax
   0x000000000040154e <+52>:    cmp    %al,%dl
   0x0000000000401550 <+54>:    je     0x401564 <bombphase_1+74>
   0x0000000000401552 <+56>:    movsbl %dl,%ecx
   0x0000000000401555 <+59>:    movsbl %al,%eax
   0x0000000000401558 <+62>:    lea    0xe(%rax),%edx
   0x000000000040155b <+65>:    mov    $0x1,%eax
   0x0000000000401560 <+70>:    cmp    %edx,%ecx
   0x0000000000401562 <+72>:    je     0x401577 <bombphase_1+93>
   0x0000000000401564 <+74>:    callq  0x401e7f <detonate_bomb_now>
   0x0000000000401569 <+79>:    mov    $0xffffffffef3450d4,%rax
   0x0000000000401570 <+86>:    jmp    0x401577 <bombphase_1+93>
   0x0000000000401572 <+88>:    mov    $0x0,%eax
   0x0000000000401577 <+93>:    add    $0x8,%rsp
   0x000000000040157b <+97>:    retq
End of assembler dump.
```

→ reserve 8 byte

check first argument, rdi

if: 0x4d → 'M'

Check if: 0x3f = (rdi+2) = '?'

each char taken 2 byte

if: 0x6a = (rdi+4) = 'j'

if 0x79 = (rdi+3) = 'y'

if 0x23 = (rdi+1) = '#'

for advance solution chang '#' to '$'.

if %sil = (rdi+15)

Value

'O' → 21st (20 index)

'A' → 37th (36 index)

M$?yj****************O**{************A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
M	$	?	Y	j											*				

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
	O			{													A		

Here's a **brief explanation** of the assembly instructions used in **bombphase_1** with easy-to-understand descriptions and examples:

1. **`sub` (Subtract)**
- **Description**: This instruction subtracts a value from a register, typically used to adjust the stack pointer (`%rsp`) to allocate space on the stack.
- **Example**:
```assembly
sub $0x8, %rsp
```
- **What it does**: Subtracts 8 from `%rsp`, creating space on the stack (like reserving a spot for temporary data).

2. **`cmpb` and `cmp` (Compare)**
- **Description**: Compares two values. It doesn't store the result, but sets flags (like equal, greater, or less) for a conditional jump that follows.
  - **`cmpb`**: Compares single bytes.
  - **`cmp`**: Compares larger values (such as words or double words).
- **Example**:
```assembly
cmpb $0x4d, (%rdi)
cmp %edx, %ecx
```
- **What it does**: Compares the first byte of the input (`%rdi`) with the value `0x4d` (ASCII `'M'`).
- In the second line, it compares two registers, `%edx` and `%ecx`.

3. **`jne` and `je` (Jump if Not Equal / Jump if Equal)**
- **Description**: These are conditional jumps based on the results of a comparison (`cmp` or `cmpb`).
  - **`jne`**: Jumps if the comparison result was **not equal**.
  - **`je`**: Jumps if the comparison result was **equal**.
- **Example**:
```assembly
cmpb $0x4d, (%rdi)
jne 0x401564
```
- **What it does**: Compares the first byte of the input to `0x4d`. If they are not equal, it jumps to the address `0x401564` (bomb detonation).

4. **`movzbl` and `mov` (Move)**
- **Description**: Moves data from one place (a register or memory location) to another.
  - **`movzbl`**: Moves a byte from memory to a register and zero-extends it (fills higher bytes with zeros).
  - **`mov`**: Moves data without any modification.
- **Example**:
```assembly
movzbl 0x14(%rdi), %edx
mov %eax, %ebx
```
- **What it does**: Moves the 21st byte from the input (located at `0x14(%rdi)`) into `%edx` and fills the rest with zeros. In the second line, it simply copies the value in `%eax` to `%ebx`.

5. **`lea` (Load Effective Address)**
- **Description**: This instruction calculates an address or a value and loads it into a register. It's often used for arithmetic operations.
- **Example**:
```assembly
lea 0xe(%rax), %edx
```
- **What it does**: Adds `0xe` (14 in decimal) to the value in `%rax` (the 37th character) and stores the result in `%edx`.

6. **`callq` (Call)**
- **Description**: Calls a function. This saves the current instruction pointer (so it can return later) and jumps to the function's address.
- **Example**:
```assembly
callq 0x401e7f
```
- **What it does**: Calls the function `detonate_bomb_now` (which detonates the bomb).

7. **`retq` (Return)**
- **Description**: Returns from a function to the point where it was called.
- **Example**:
```assembly
retq
```
- **What it does**: Returns from the current function, signaling success (bomb defused).

Summary:
- **`cmp`/`cmpb`**: Compares two values.
- **`sub`**: Reserves stack space by subtracting from `%rsp`.
- **`jne`/`je`**: Conditional jumps based on the comparison result.
- **`mov`/`movzbl`**: Moves data between registers or memory.
- **`lea`**: Used for address calculations and arithmetic.
- **`callq`**: Calls a function.
- **`retq`**: Returns from a function.

Linux procedure call conventions
Parameters: %rdi, %rsi, %rdx, %rcx, %r8, %r9.

Additional parameters are passed on the stack.
Return value: %rax
Caller-save registers:
Parameters and %rax, %r10, %r11
Callee-save registers: %rbx, %rbp, %r12, %r13, %r14, %r15
Special: %rsp

```
Dump of assembler code for function bombphase_2:
   0x000000000040163a <+0>:     push   %rbx
   0x000000000040163b <+1>:     sub    $0x10,%rsp
   0x000000000040163f <+5>:     mov    %rdi,%rbx
   0x0000000000401642 <+8>:     movabs $0x656d6f6d72656874,%rax
   0x000000000040164c <+18>:    mov    %rax,(%rsp)
   0x0000000000401650 <+22>:    movl   $0x726574,0x8(%rsp)
   0x0000000000401658 <+30>:    mov    $0xb,%edx
   0x000000000040165d <+35>:    mov    %rsp,%rsi
   0x0000000000401660 <+38>:    callq  0x4015c0 <my_strncmp>
   0x0000000000401665 <+43>:    test   %eax,%eax
   0x0000000000401667 <+45>:    jne    0x401687 <bombphase_2+77>
   0x0000000000401669 <+47>:    mov    $0x0,%eax
   0x000000000040166e <+52>:    cmpb   $0x0,0xb(%rbx)
   0x0000000000401672 <+56>:    je     0x401693 <bombphase_2+89>
   0x0000000000401674 <+58>:    mov    %rbx,%rdi
   0x0000000000401677 <+61>:    callq  0x40161c <my_strlen>
   0x000000000040167c <+66>:    cmp    $0x1a,%eax
   0x000000000040167f <+69>:    sete   %al
   0x0000000000401682 <+72>:    movzbl %al,%eax
   0x0000000000401685 <+75>:    jmp    0x401693 <bombphase_2+89>
   0x0000000000401687 <+77>:    callq  0x401e7f <detonate_bomb_now>
   0x000000000040168c <+82>:    mov    $0xffffffffef3450d4,%rax
   0x0000000000401693 <+89>:    add    $0x10,%rsp
   0x0000000000401697 <+93>:    pop    %rbx
   0x0000000000401698 <+94>:    retq
End of assembler dump.
```

'thermome' → RSP

'ter' → (RSP+8)

→ store '11' in edx

- The **first argument** goes in %rdi (input string).
- The **second argument** goes in %rsi (string "thermometer").
- The **third argument** goes in %edx (length, 11).

→ return = 0

→ not-null check at 12th char.

→ simple solution.

→ input (rdi) which was set 3rd line.

→ check what in the len?

26 = eax?

→ al = 1

1. **`push` (Push)**
- **Description**: Pushes a value onto the stack. This instruction is used to save the current value of a register so it can be restored later.
- **Example**:
```assembly
push %rbx
```
- **What it does**: Saves the current value of `%rbx` onto the stack. This is typically done at the start of a function to preserve the caller's value of `%rbx`.

2. **`movabs` (Move Absolute)**
- **Description**: Moves an absolute value (usually an immediate constant that doesn't fit into 32 bits) into a register. This is used when you need to load a 64-bit value into a register.
- **Example**:
```assembly
movabs $0x656d6f6d72656874, %rax
```
- **What it does**: Loads the 64-bit constant value `0x656d6f6d72656874` (which represents the ASCII for part of the string "thermometer") into the `%rax` register.

3. **`movl` (Move Long)**
- **Description**: Moves a 32-bit value (long) from one location to another. It operates on 4-byte (32-bit) data.
- **Example**:
```assembly
movl $0x726574, 0x8(%rsp)
```
- **What it does**: Moves the 32-bit value `0x726574` (the ASCII for "ret", completing the "thermometer" string) into the stack memory at an offset of 8 bytes from the top of the stack (`%rsp`).

4. **`test` (Test)**
- **Description**: Performs a bitwise AND between two values and sets CPU flags based on the result, but it doesn't store the result. It's typically used to check for zero or non-zero values.
- **Example**:
```assembly
test %eax, %eax
```
- **What it does**: ANDs the value in `%eax` with itself to check if it is zero. This sets flags that are used by subsequent conditional instructions like `je` or `jne`.

5. **`sete` (Set if Equal)**
- **Description**: Sets the value of a register or memory location to 1 if the zero flag (ZF) is set (meaning the previous comparison was equal), or 0 otherwise.
- **Example**:
```assembly
sete %al
```
- **What it does**: Sets the `%al` register to 1 if the previous comparison indicated equality, otherwise it sets it to 0.

6. **`movzbl` (Move Zero-Extend Byte to Long)**
- **Description**: Moves a byte from memory or a register to a register and zero-extends it, filling the remaining bits with zeroes.
- **Example**:
```assembly
movzbl %al, %eax
```
- **What it does**: Moves the byte value in `%al` (lower 8 bits of `%rax`) to the 32-bit `%eax` register and zero-extends the remaining bits (filling them with zeroes).

7. **`add` (Add)**
- **Description**: Adds a value to a register or memory location and stores the result in the destination.

~~~
  - **Example**:
    ```assembly
    add $0x10, %rsp
    ```
    - **What it does**: Adds 16 to `%rsp`, effectively restoring the stack pointer after space was previously allocated (cleaning up the stack).

### 8. **`pop` (Pop)**
  - **Description**: Pops a value from the stack into a register, typically used to restore a saved value.
  - **Example**:
    ```assembly
    pop %rbx
    ```
    - **What it does**: Restores the value of `%rbx` from the top of the stack, undoing the effect of the earlier `push %rbx`.

### Summary of Instructions:
- **`push`**: Saves a register value on the stack.
- **`movabs`**: Moves a 64-bit immediate value into a register.
- **`movl`**: Moves a 32-bit value to a memory location.
- **`test`**: Tests values (sets flags but doesn't store the result).
- **`sete`**: Sets a value to 1 if the previous comparison was equal.
- **`movzbl`**: Moves a byte and zero-extends it to 32 bits.
- **`add`**: Adds to a register or memory location.
- **`pop`**: Restores a register value from the stack.
~~~

Stop in these two address:

0x0000000000401594 for X
0x00000000004015a7 for Y

```
(gdb) break *0x0000000000401599
(gdb) break *0x00000000004015a7
(gdb) break *0x00000000004015aa
```

I have to provide 4 input(rdi =4)

- 445 is stored in %rsi (first input).
- 264 is likely stored in %rdx (second input).
- 181 is stored in %rcx (third input).
- -428 is stored in %r8 (fourth input)

445 264 181 -428

```
Dump of assembler code for function bombphase_3:
   0x000000000040157c <+0>:     sub    $0x8,%rsp
   0x0000000000401580 <+4>:     cmp    $0x4,%rdi
   0x0000000000401584 <+8>:     je     0x401594 <bombphase_3+24>
   0x0000000000401586 <+10>:    callq  0x401e7f <detonate_bomb_now>
   0x000000000040158b <+15>:    mov    $0xffffffffef3450d4,%rax
   0x0000000000401592 <+22>:    jmp    0x4015bb <bombphase_3+63>
   0x0000000000401594 <+24>:    mov    $0x1bd,%eax
   0x0000000000401599 <+29>:    sub    %rsi,%rax
   0x000000000040159c <+32>:    not    %rax
   0x000000000040159f <+35>:    and    $0x11e,%eax
   0x00000000004015a4 <+40>:    not    %r8
   0x00000000004015a7 <+43>:    xor    %r8,%rax
   0x00000000004015aa <+46>:    cmp    %rcx,%rax
   0x00000000004015ad <+49>:    jne    0x401586 <bombphase_3+10>
   0x00000000004015af <+51>:    cmp    $0xfffffffffffffe54,%rax
   0x00000000004015b5 <+57>:    sete   %al
   0x00000000004015b8 <+60>:    movzbl %al,%eax
   0x00000000004015bb <+63>:    add    $0x8,%rsp
   0x00000000004015bf <+67>:    retq
End of assembler dump.
```

decrened 8 byte

rdi ⇒ 4 : have to give 4 input.

→ detonate

→ rax ⇒ ffffs (Just to confuse)

→ eax ⇒ 445

→ rax -= rsi ⇒ 445-445 = 0 → 1

→ invert

→ eax = 286

-428 → 427

→ 427 XOR 286 → 181 (rax)

→ 181 and 181 → equal

detonate

→ rax = -428

~~~
### 1. **`not` (Bitwise NOT)**
  - **Description**: Performs a bitwise NOT operation, which flips all the bits in the specified register. This means that `0`s become `1`s and `1`s become `0`s.
  - **Example**:
    ```assembly
    not %rax
    ```
    - **What it does**: Flips all the bits in the `%rax` register.

### 2. **`and` (Bitwise AND)**
  - **Description**: Performs a bitwise AND between two operands. For each bit, the result is `1` only if both bits in the operands are `1`, otherwise the result is `0`.
  - **Example**:
    ```assembly
    and $0x11e, %eax
    ```
    - **What it does**: Performs a bitwise AND between the value in `%eax` and the constant `0x11e`. This effectively filters certain bits in `%eax`, based on the mask `0x11e`.

### 3. **`xor` (Bitwise XOR)**
  - **Description**: Performs a bitwise XOR (exclusive OR) between two operands. For each bit, the result is `1` if the bits in the two operands are different, and `0` if they are the same.
  - **Example**:
    ```assembly
    xor %r8, %rax
    ```
    - **What it does**: XORs the values in `%r8` and `%rax`, storing the result in `%rax`.
~~~

→ input

rax = eax = ax = ~~2078~~ = 2356

rdi = ~~input~~ = 7629261

stack 808 byte

→ rsi = rdi

```
Dump of assembler code for function bombphase_4:
   0x000000000040172c <+0>:     sub    $0x328,%rsp
   0x0000000000401733 <+7>:     mov    %rdi,%rsi
   0x0000000000401736 <+10>:    movb   $0xd,0x24(%rsp)
   0x000000000040173b <+15>:    movw   $0x1fa7,0x26(%rsp)
   0x0000000000401742 <+22>:    movq   $0x7b3fd988,0x30(%rsp)
   0x000000000040174b <+31>:    movw   $0x0,0x28(%rsp)
   0x0000000000401752 <+38>:    movl   $0x1fd5d509,0x20(%rsp)
   0x000000000040175a <+46>:    movw   $0x81e,0x1c(%rsp)
   0x0000000000401761 <+53>:    movzwl 0x1c(%rsp),%eax
   0x0000000000401766 <+58>:    add    $0x116,%ax
   0x000000000040176a <+62>:    mov    %ax,0x1c(%rsp)
   0x000000000040176f <+67>:    movl   $0x74656d,0x10(%rsp)
   0x0000000000401777 <+75>:    lea    0x10(%rsp),%rdi
   0x000000000040177c <+80>:    callq  0x4014e4 <strcat>
```

→ 13

→ 8103

→ (28+rsp) = 2078

value only → eax = 2078

→ ax = 278+2078 = 2356

→ (28+rsp) =

→ rdi = 0x74656d + em

→ input replaced cause a new func will be called.

rsp =

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
															*				

2078

0x74656d

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
				13		8103		0											

533429001

~~2078~~ 2356

2069695880

rdx=0x7b3fd788

'met'

10rsp → 'met'

rdi = 0x10rsp = 'met'

rsi = input

```
Dump of assembler code for function strcat:
   0x00000000004014e4 <+0>:     mov    %rdi,%rax
   0x00000000004014e7 <+3>:     cmpb   $0x0,(%rdi)
   0x00000000004014ea <+6>:     je     0x4014fa <strcat+22>
   0x00000000004014ec <+8>:     mov    %rdi,%rdx
```

→ rax = 'met'

→ rdi = 0 ? not equal (value)

→ rdx = 'met'] runn loop to

```
Dump of assembler code for function strcat:
   0x00000000004014e4 <+0>:     mov    %rdi,%rax
   0x00000000004014e7 <+3>:     cmpb   $0x0,(%rdi)
   0x00000000004014ea <+6>:     je     0x4014fa <strcat+22>
   0x00000000004014ec <+8>:     mov    %rdi,%rdx
   0x00000000004014ef <+11>:    add    $0x1,%rdx
   0x00000000004014f3 <+15>:    cmpb   $0x0,(%rdx)
   0x00000000004014f6 <+18>:    jne    0x4014ef <strcat+11>
   0x00000000004014f8 <+20>:    jmp    0x4014fd <strcat+25>
   0x00000000004014fa <+22>:    mov    %rdi,%rdx
   0x00000000004014fd <+25>:    movzbl (%rsi),%ecx
   0x0000000000401500 <+28>:    test   %cl,%cl
   0x0000000000401502 <+30>:    je     0x401516 <strcat+50>
   0x0000000000401504 <+32>:    add    $0x1,%rdx
   0x0000000000401508 <+36>:    add    $0x1,%rsi
   0x000000000040150c <+40>:    mov    %cl,-0x1(%rdx)
   0x000000000040150f <+43>:    movzbl (%rsi),%ecx
   0x0000000000401512 <+46>:    test   %cl,%cl
   0x0000000000401514 <+48>:    jne    0x401504 <strcat+32>
   0x0000000000401516 <+50>:    movb   $0x0,(%rdx)
   0x0000000000401519 <+53>:    retq
End of assembler dump.
```

rax = 'met'

rdi = 0 ? not ean

rdx = 'met'

met null
0 1 2 3

runs loop to find null in

ecx = input

since it has values, it would never be zero "0".

rdx in already = " "

input string + 1

Now:etAAAAAAAAAAAAAAAAA;3!
Was:metAAAAAAAAAAAAAAAAA;3!

```
(gdb) x/s $rdx
0x7fffffffdbc4: "duzzaman4\tha\t\325\325\037\r2\247\037"
(gdb) x/s $rax
0x7fffffffdbc0: "metmduzzaman4\tha\t\325\325\037\r2\247\037"
```

```
  0x0000000000401781 <+85>:    mov    0x30(%rsp),%rdx
  0x0000000000401786 <+90>:    mov    $0x0,%eax
  0x000000000040178b <+95>:    cmp    $0x7b3fd988,%rdx
  0x0000000000401792 <+102>:   je     0x4017a0 <bombphase_4+116>
  0x0000000000401794 <+104>:   callq  0x401e7f <detonate_bomb_now>
  0x0000000000401799 <+109>:   mov    $0xffffffffef3450d4,%rax
  0x00000000004017a0 <+116>:   movzbl 0x24(%rsp),%edx
  0x00000000004017a5 <+121>:   cmp    $0x3b,%dl
  0x00000000004017a8 <+124>:   je     0x4017b6 <bombphase_4+138>
  0x00000000004017aa <+126>:   callq  0x401e7f <detonate_bomb_now>
  0x00000000004017af <+131>:   mov    $0xffffffffef3450d4,%rax
  0x00000000004017b6 <+138>:   movzwl 0x28(%rsp),%edx
  0x00000000004017bb <+143>:   test   %dx,%dx
  0x00000000004017be <+146>:   je     0x4017cc <bombphase_4+160>
  0x00000000004017c0 <+148>:   callq  0x401e7f <detonate_bomb_now>
  0x00000000004017c5 <+153>:   mov    $0xffffffffef3450d4,%rax
  0x00000000004017cc <+160>:   movzwl 0x26(%rsp),%edx
  0x00000000004017d1 <+165>:   cmp    $0x2133,%dx
  0x00000000004017d6 <+170>:   je     0x4017e4 <bombphase_4+184>
  0x00000000004017d8 <+172>:   callq  0x401e7f <detonate_bomb_now>
  0x00000000004017dd <+177>:   mov    $0xffffffffef3450d4,%rax
  0x00000000004017e4 <+184>:   movzwl 0x1c(%rsp),%edx
  0x00000000004017e9 <+189>:   mov    0x20(%rsp),%ecx
  0x00000000004017ed <+193>:   movswl %dx,%edx
  0x00000000004017f0 <+196>:   shl    $0x5,%edx
  0x00000000004017f3 <+199>:   add    %ecx,%edx
  0x00000000004017f5 <+201>:   cmp    $0x4141e54c,%edx
  0x00000000004017fb <+207>:   sete   %dl
  0x00000000004017fe <+210>:   movzbl %dl,%edx
  0x0000000000401801 <+213>:   or     %rdx,%rax

  0x0000000000401804 <+216>:   add    $0x328,%rsp
  0x000000000040180b <+223>:   retq
End of assembler dump.
```

rdx = 0x7b3fd988

eax=0

match -> jmp 116

0x24(%rsp) = 0xd

6

95
121
143
165
201

```
Break bombphase_4
(gdb) break *0x40178b
0x0000000000401794
(gdb) break *0x4017a5
0x00000000004017aa
(gdb) break *0x4017bb
0x00000000004017c0
(gdb) break *0x4017d1
0x00000000004017d8
(gdb) break *0x4017f5
```