Lab 1: Defusing a Binary Bomb

Sungmin Kim and Sangjin Ha (snu.comarch.2016@gmail.com)
Architecture & Code optimization (ARC) Lab

October 7th, 2016

Overview

- **The goal of this lab**
  - To defuse a binary bomb using correct strings
- **By finishing this lab successfully**
  - You will become familiar with **x86-64 assembly language**
  - You will become familiar with a **debugger**
- **You need a Linux environment to do this lab**
  - You can access a Linux machine in “Software Lab” at Building 302 (Room 311-1) with your own ID and PW
  - You can also install Linux on your PC using VirtualBox (See Appendix B)

Step 1: Get your bomb

- **Go to our webpage (http://arc.snu.ac.kr:54321)**
  - Your student ID (e.g. 2016-12345) and email address

CS:APP Binary Bomb Request

Fill in the form and then click the Submit button.

Hit the Reset button to get a clean form.

Legal characters are spaces, letters, numbers, underscores ('_'), hyphens ('-'), at signs ('@'), and dots ('.').

User name
Enter your student ID (e.g. 2016-12345)

Email address

Submit Reset

- **Then, you can get a .tar file, "bomb*k*.tar"**
  - README: Identifies the bomb and its owners
  - bomb: The executable binary bomb
  - bomb.c: Source file with the bomb's main routine

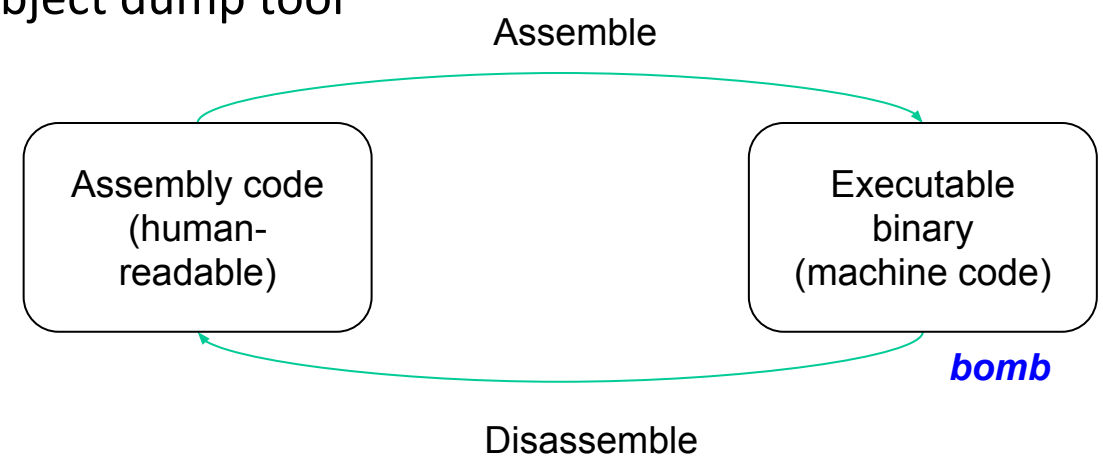
Step 2: Defuse your bomb

- **You need to find correct strings for defusing your bomb**
  - Using a debugger and/or other tools

- **Your bomb has 6 phases**
  - Each phase requires you to type the correct string to pass
  - If your input is wrong, your bomb will explode
  - You must pass all 6 phases to defuse the bomb!

Step 2: Defuse your bomb

- **You only know the C code of the main routine: *bomb.c***
- **To defuse your bomb, you need to look inside the binary file (named "bomb")**
  - By disassembling it
- **You should check out disassembled code**
  - Using debugger or
  - Using object dump tool



Step 2: Defuse your bomb

- **Using objdump (disassembler)**

```
$> objdump -t bomb
```

  - This command will print out the names of all functions and global variables, all the functions the bomb calls, and their addresses

```
$> objdump -d bomb
```

  - Use this command to disassemble all of the code in the bomb
  - You can also take a look at individual functions

- **Using gdb (debugger)**
  - See on **Appendix A** for details
  - More powerful!

Grading guideline

- **Each time your bomb explodes, it will notify our server.**
- **You can get points by defusing each phase.**
  - Phase 1-4: 15 points/phase
  - Phase 5-6: 20 points/phase
  - Total 100 points
- **You will lose points when your bomb explodes**
  - -1 points per two explosion (maximum -20 points)
- **Due date is Oct 21st (Fri) 09:59 am before the class**
  - We will shut down the grading server on time

Grading guideline

- **Your score can be found at http://arc.snu.ac.kr:54321/scoreboard**
  - This webpage is updated every 30 seconds
  - Check out your bomb (identified by your bomb #)
- **You can download a new bomb repeatedly using the same ID**
  - Regardless of how many bombs your defuse, your maximum point will still be 100.

Bomb Lab Scoreboard

This page contains the latest information that we have received from your bomb. If your solution is marked **invalid**, this means your bomb reported a solution that didn't actually defuse your bomb.

Last updated: Thu Oct 6 18:27:56 2016 (updated every 30 secs)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb1	Thu Sep 29 17:46	7	1	100	valid
2	bomb4	Thu Oct 6 16:30	7	16	92	valid
3	bomb3	Thu Sep 29 20:23	0	1	0	valid
4	bomb2	Thu Sep 29 18:29	0	2	-1	valid

Summary [phase:cnt] [1:0] [2:0] [3:0] [4:0] [5:0] [6:0] [7:2] total defused = 2/4

Q&A

Appendix A

- How to use GNU debugger

How to use GNU debugger

- **Using GNU debugger (=GDB)**
  - You can see what is going on inside a program while it is running
  - You can start your program, specifying anything that might affect its behavior
  - You can make your program stop under a specified condition
  - You can examine what has happened (i.e., the program's state), when your program has stopped
  - Can change the value of variable in your program

How to use GNU debugger

- **Install GDB**

```
$> sudo apt-get install gdb
```

- **Run executable with GDB**

```
$> gdb nameOfExecutable
```

  - In this lab, “nameOfExecutable” is ‘**bomb**’

How to use GNU debugger

- **Basic instructions**
  - `(gdb) run` : Start the program
  - `(gdb) continue` : Run the program until next breakpoint
  - `(gdb) breakpoint` : Make a breakpoint
  - `(gdb) delete` : Delete a breakpoint
  - `(gdb) step` : Run next line of code
  - `(gdb) next` : Run next line of code (not jumping into a function)
  - `(gdb) quit` : Quit gdb
- **Instructions for assembly code**
  - `(gdb) disassemble` : Disassemble the function / lines of code
  - `(gdb) stepi` : Run next line of assembly code

How to use GNU debugger

- **Variable print instruction**
  - (gdb) `print` *`func`* : Print address of function *func*
  - (gdb) `p` *`var`* : Print value of variable *var*
  - (gdb) `p/[format]` *`var`* : Print value of *var* with format
    - Format: t = binany, o = octal, d = int, u = unsigned int, x = hexadecimal, c = char, f = floating-point
- **Memory print instruction**
  - (gdb) `x/[range][format][unit]` *`addr`* : Print memory value
    - Format: t = binany, o = octal, d = int, u = unsigned int, x = hexadecimal, c = char, f = floating-point, **s = string**, **i = assembly instr**
    - Unit: b = byte, h = halfword (2-byte), w = word (4-byte), g = giant word (8-byte)

How to use GNU debugger

- **Information print instruction**
  - (gdb) `info registers` : Print all registers' value
  - (gdb) `info breakpoints` : Print all breakpoints

How to use GNU debugger

- **If you need more instruction detail**

```
$> man gdb
```

  - `(gdb) help`

- **References**
  - **http://visualgdb.com/gdbreference/commands/**
  - http://www.yolinux.com/TUTORIALS/GDB-Commands.html
  - 유닉스 리눅스 프로그래밍 필수 유틸리티, 백창우, 한빛미디어

Appendix B

- How to set up a Linux environment

Option 1: Windows' bash shell

- **It can run Linux command-line utilities on Windows 10 (64bit only)**
- **It's based on Ubuntu**

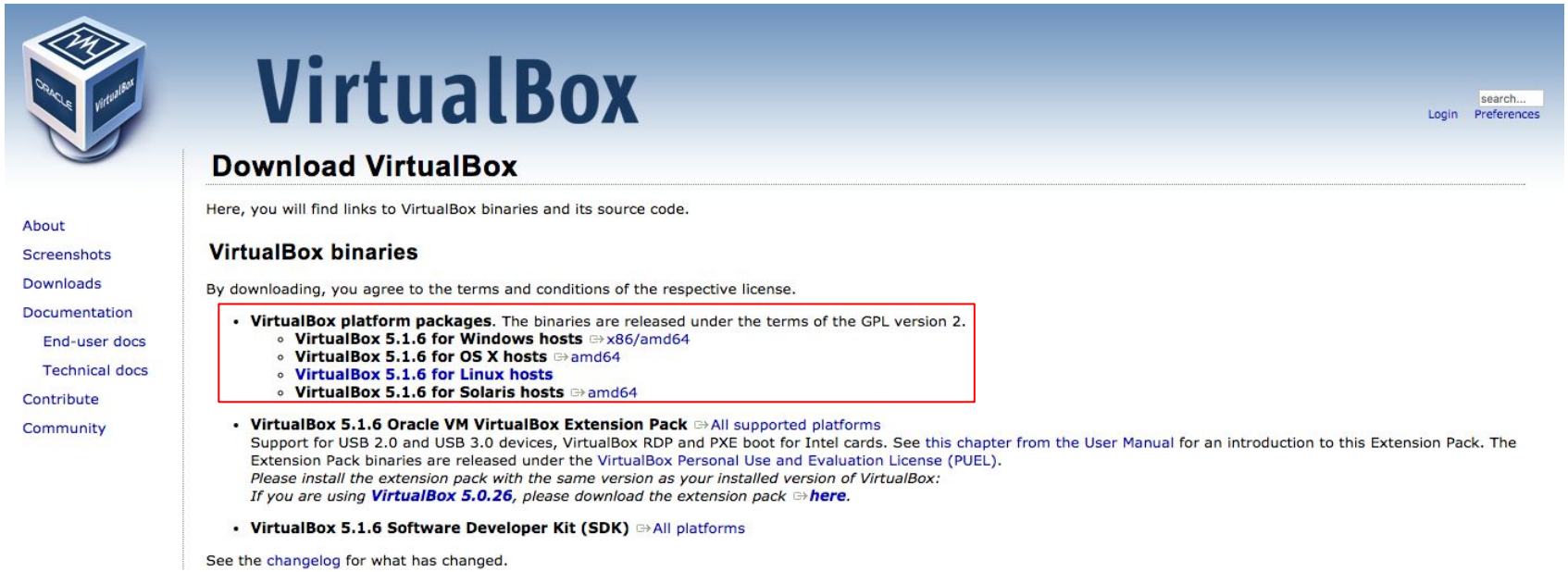
ubuntu on Windows 10

- **Here is a setup guideline**
  - http://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10/
- **Get utilities for the lab**

```
$> sudo apt-get install build-essential gdb
```

Option 2: Ubuntu on VirtualBox

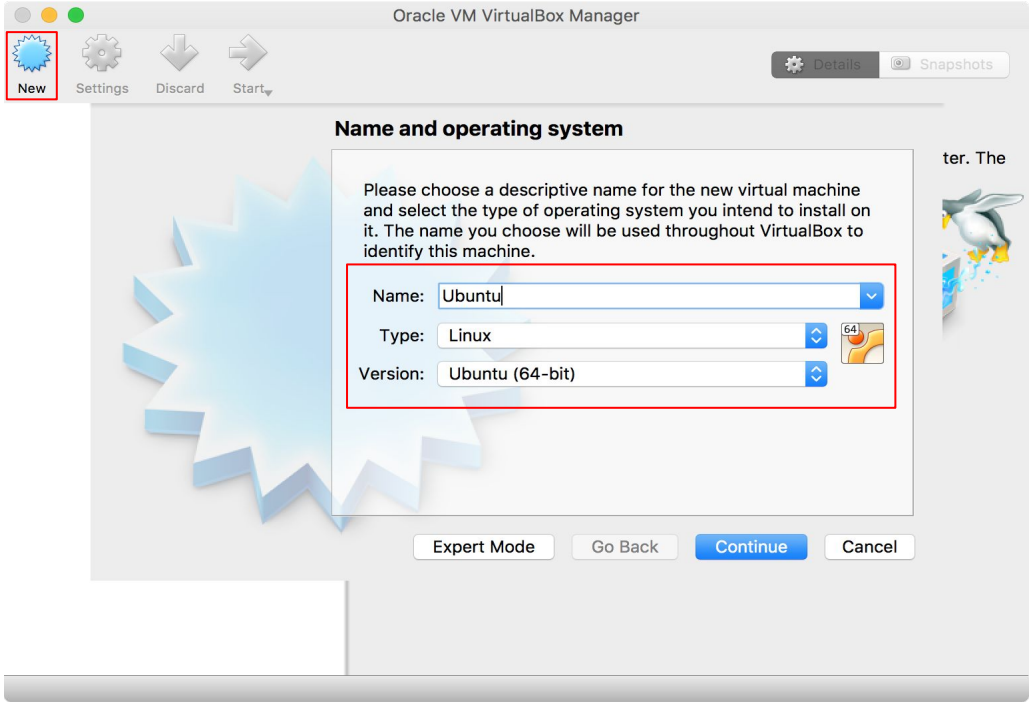
- **Download and install Oracle VirtualBox 5.1**



- **Get Ubuntu 14.04 LTS CD image**
  - http://releases.ubuntu.com/14.04/ubuntu-14.04.4-desktop-amd64.iso

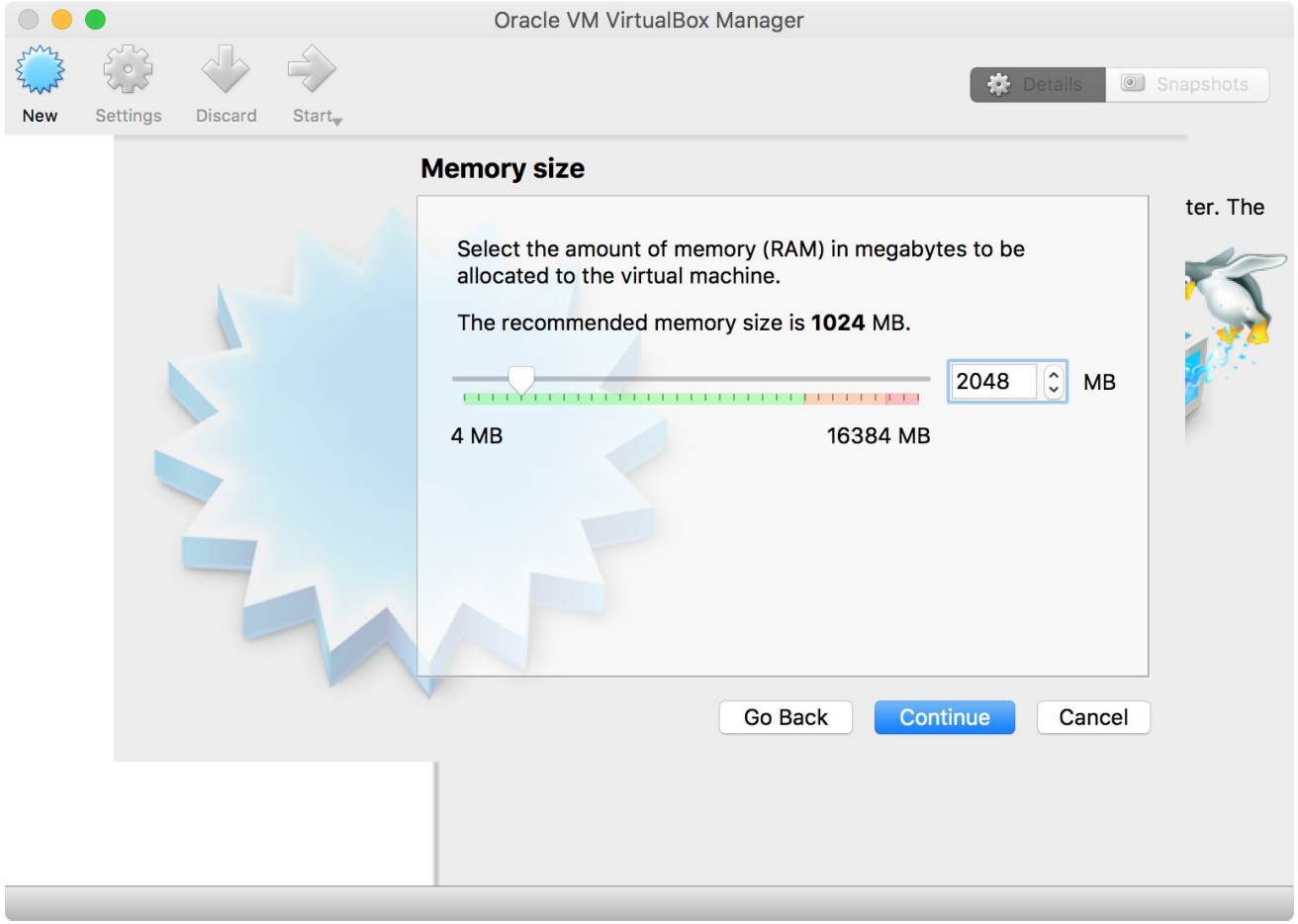
Install Ubuntu on VirtualBox

- Run VirtualBox
- Click “New”
- Type “Ubuntu” into name
- Make sure that the OS type and version are correct



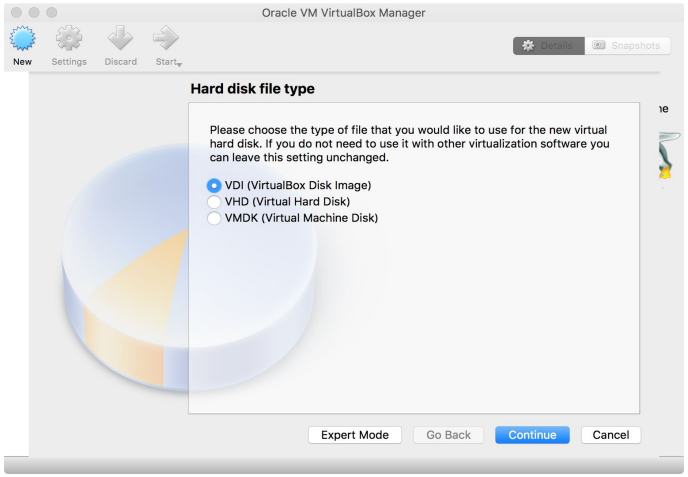
Install Ubuntu on VirtualBox

- **The recommended memory size depends on your system (default is 1GB)**



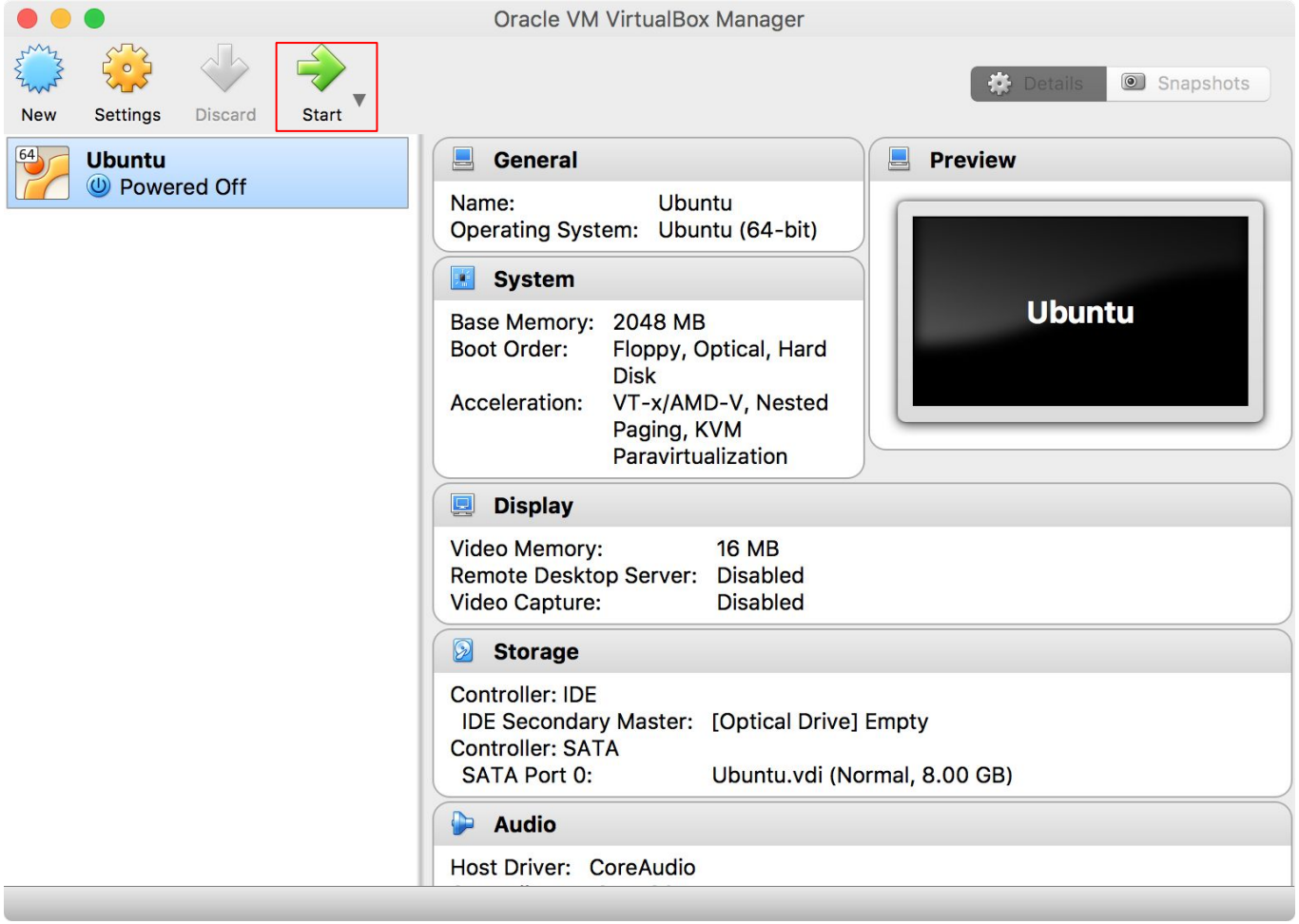
Install Ubuntu on VirtualBox

- "Create, Continue, Continue, Create"



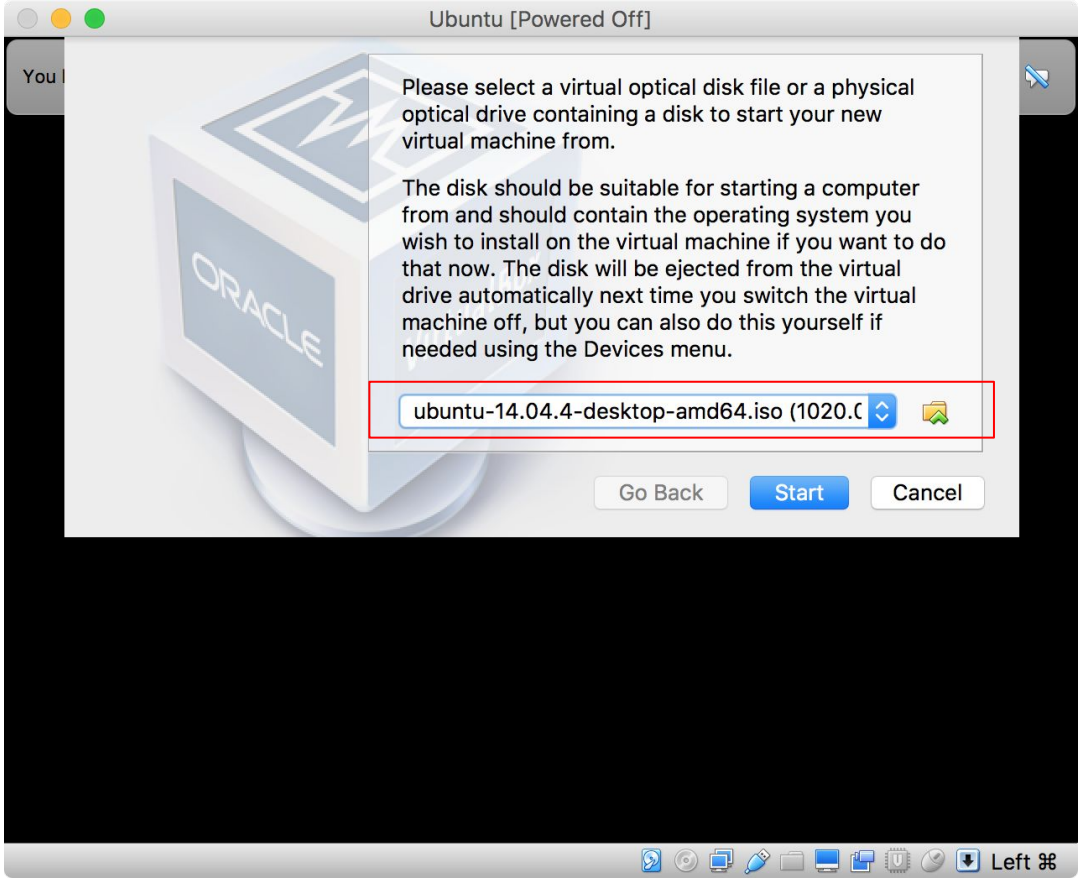
Install Ubuntu on VirtualBox

- **Now, click "Start"**



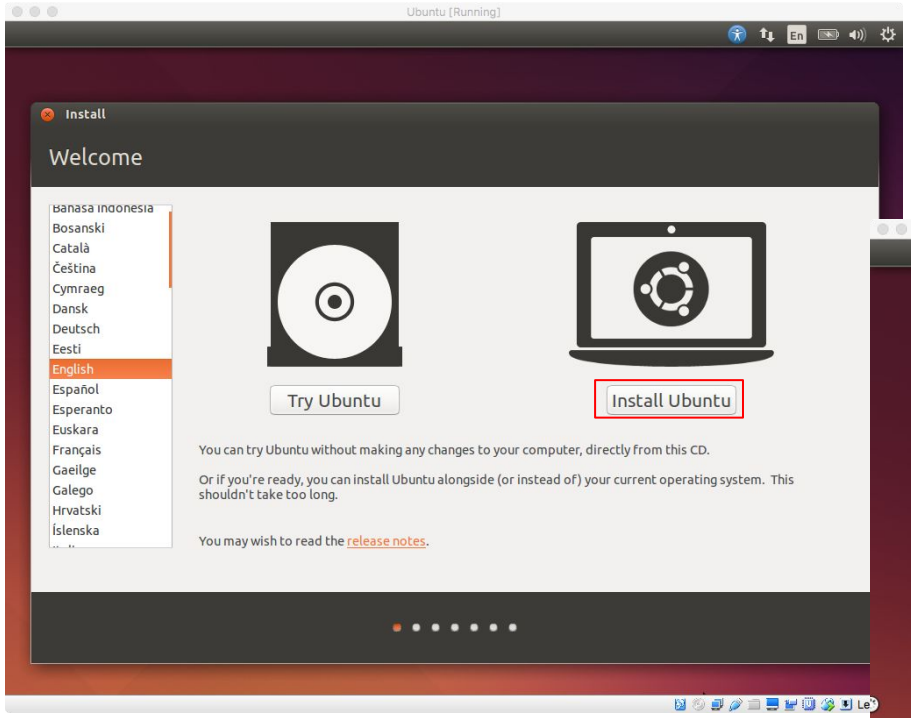
Install Ubuntu on VirtualBox

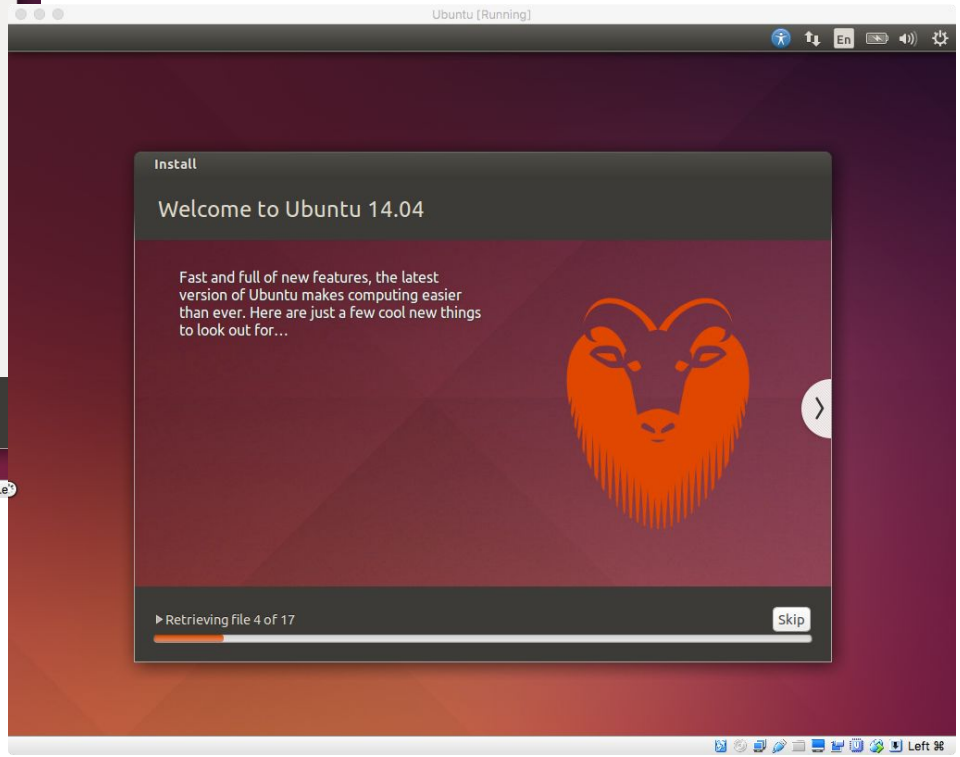
- Browse the downloaded Ubuntu 14.04 ISO image
- Click “Start” to boot



Install Ubuntu on VirtualBox

- **Install Ubuntu and now you are ready to go!**





(Optional) VirtualBox Guest Addition

- Automatic adjustment of resolution of guest OS
- Integration of mouse and keyboard