Secure Programing Labs Cheatsheet

Some useful commands

General linux

`make` Builds your code
`gcc <source-code>` C compiler
  `-o <program>` to name the executable, else it will be called `a.out`
  `-O 0` disables optimisations, giving simpler code
`objdump <program>` Lets you view information about a program
  `-d` to disassemble to assembly
  `-s` to disassemble to ASCII format
  `-j` to disassemble named section
`strace <program>` Runs the program and shows systemcalls as they are made (usefull for debugging shellcode)
`nc <host> <port>` The netcat program. Useful for sending and listening to data going between ports. By default writes.
  `-l` to listen

`ip addr` See what ip address the virtual machine has

GDB

GDB is the go-to command line debugger. For some primitive GUI support, you can try running it inside Emacs (`M-x gdb`) or with a curses UI, `gdb --tui`.

`gdb <program>` Runs GDB on your program
  `-x <gdbinit>` lets you run a script when GDB is first loaded

Inside GDB, the following commmands are useful:

`file <binary>` Start a new debugging section with the target binary.
  The current debugging section will be ended.
`run [ <args> ] [ < <input> ]` Run your program with optional args and input
  Can use backticks in the args to run an external command (such as 'perl -e 'print "A"x9001'')
`set args [ <args> ] [ < <input> ]` Specifies the args for the `run` command automatically
`awatch <address>` Sticks a read/write watch point whenever the memory at the address is accessed
`b <break-point>` Set a break point at a memory location (which can be a function name, e.g., `b main` or a de-referenced pointer, as `b *0x123456`)
`c` Continue
`si` Step instruction
`x/32x $ebp` Prints the memory as hexadecimal ints for the 32 bytes at the memory pointed to by the register `$ebp`
`x/8i $eip` Prints the memory as disassembled instructions the 8 bytes at the memory pointed to by the register `$eip`

p/x $ebp Print the value in the register `$ebp`
disas Shows the disassembly for wherever the instruction pointer is
list Shows where you are in the source code (if debugging data is on)
<!– set disassembly-flavor intel Gives you Intel-style disassembly –>
help The manual!

Radare

Radare is a *really* powerful dissembler framework It is worth learning but not required for these labs. Here are a couple of commands which help in constructing shellcode. More documentations please read the radare2book from the link below.

https://www.gitbook.com/book/radare/radare2book/details

rasm2 "nop;nop;nop" Gives you the bytecode for three `nop` instructions
rasm2 -f <file> -a x86.as -C Gives you the bytecode for assembler instructions in `<file>`, avoiding zeros and printing the result as a C-formatted string
rabin2 -z <binary file> Print all string address in the memory