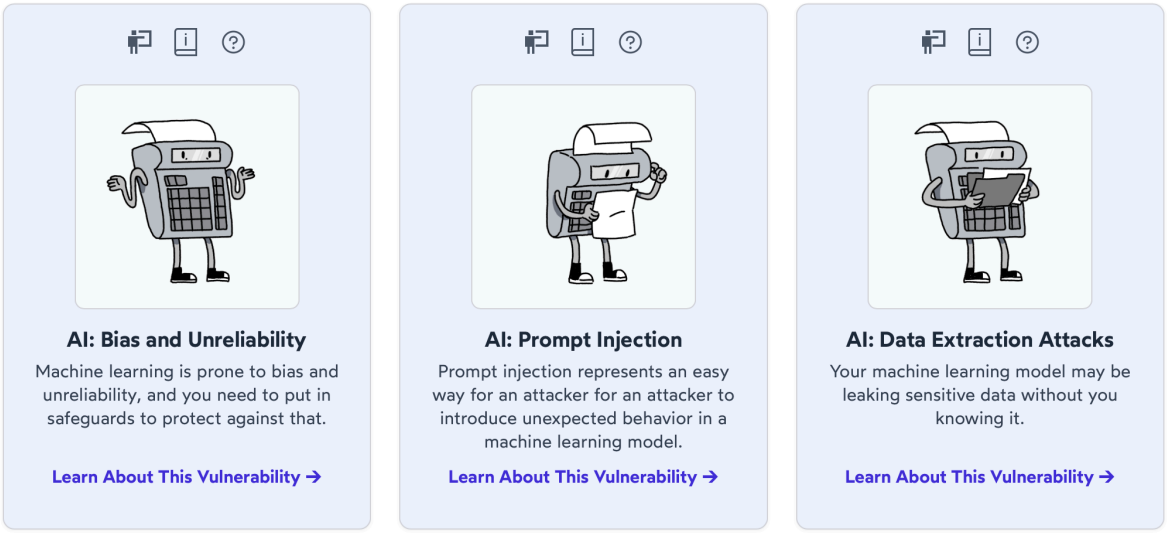
COM320 Lab – Hacksplaining & Linux Introduction

Professor Kevin Curran

Contents

1. Hacksplaining

1. Visit https://www.hacksplaining.com/lessons (register for free account if needed)



2. Complete the three AI vulnerabilities which are **AI: Bias and Unreliability, AI: Prompt Injection** and **AI: Data Extraction Attacks.**

2. Linux Environment Basics

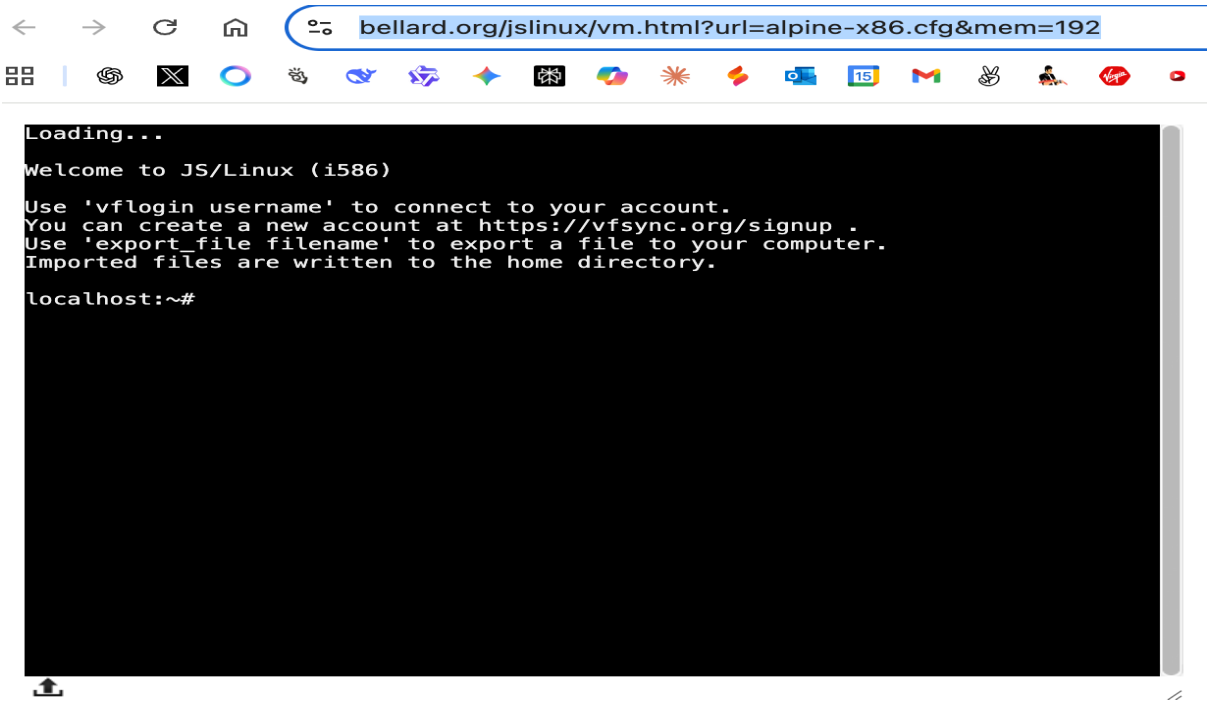
This part prepares you for some of the modules to come, which heavily rely on proficiency with the basic usage of Linux.

Lab Objectives:

1. Overview of Linux including basic file manipulation and movement within the shell.
2. Basic proficiency of the Linux Bash Shell, Text manipulation and Bash Shell scripting.
3. Knowledge of Cross-Site Scripting

2.1 Finding your way around Linux

1. Launch **Linux** from by visiting https://bellard.org/jslinux/vm.html?url=alpine-x86.cfg&mem=192
   .



.

You should now be logged into a 'portable' free online Linux version called Alpine. Alpine Linux is an independent, non-commercial, general purpose Linux distribution designed for power users who appreciate security, simplicity and resource efficiency.

Linux is an open-source operating system first developed by Linus Torvalds in 1991. Unlike proprietary systems like Windows or macOS, Linux's source code is freely available, allowing anyone to modify and distribute it. This openness has led to a vast ecosystem of distributions (or "distros") like Ubuntu, Fedora, and Alpine, each tailored to specific needs. Linux powers a wide range of devices, from servers and supercomputers to smartphones and embedded systems.

Its popularity comes from several strengths: it is free, highly customisable, and known for stability and security. The global community of developers continuously improves it, making it a reliable choice for both hobbyists and enterprises. Linux also excels in performance, running efficiently on everything from old hardware to cutting-edge systems, which explains its widespread adoption across industries.

2.2 Linux Basic & Linux Services

The default shell used by Alpine Linux is the BusyBox variant of the ash shell. A shell is a computer program that serves as an interface for users to interact with an operating system, allowing them to execute commands and control the computer's functions. While the term "shell" can also refer to the hard outer covering of an animal, nut, or egg, in the context of computing, it is the layer of software that provides access to the operating system's services, either through a command-line interface (CLI) or a graphical user interface (GUI).

BusyBox is a command processor (shell) that typically runs in a text window where the user types commands that cause actions. The shell can also read and execute commands from a file, called a shell script. Like all Unix shells, it supports filename globbing (wildcard matching), piping, here documents, command substitution, variables, and control structures for condition-testing and iteration. The keywords, syntax and other basic features of the language are all copied from sh. Other features, e.g., history, are copied from csh and ksh.

The following part will cover some of the basic tools we will be working with regularly - proficiency with them will be assumed. The shell (or any other shell for that matter) is a very powerful scripting environment. On many occasions we need to automate an action or perform repetitive time-consuming tasks. This is where bash scripting comes in handy.

2.2.1 Linux basic commands

Type the commands in red & black below. The blue text with symbol # is simply comments on some commands to show you what is happening. Just type the commands in red.

pwd *# print working directory*

```
localhost:~# pwd
/root
localhost:~#
```

mkdir mydir *# make a new directory, mydir*

```
localhost:~# pwd
/root
localhost:~# mkdir mydir
localhost:~#
```

cd mydir *# move into the /mydir directory*

```
localhost:~# cd mydir
localhost:~/mydir#
```

pwd *# now you are in ~/mydir*

```
localhost:~/mydir# pwd
/root/mydir
localhost:~/mydir#
```

touch myfile *# create a blank file called myfile*

```
localhost:~/mydir# touch myfile
localhost:~/mydir#
```

ls myfile *# list any file with name 'myfile'*

```
localhost:~/mydir# ls myfile
myfile
localhost:~/mydir#
```

ls -alrth myfile *# list metadata on myfile*

```
localhost:~/mydir# ls -alrth myfile
-rw-r--r--    1 root     root           0 Sep 15 16:30 myfile
localhost:~/mydir#
```

echo "line1" >> myfile **# append via '>>' to a file**

```
localhost:~/mydir# echo "line1" >>myfile
localhost:~/mydir#
```

cat myfile **# Display the contents of myfile**

```
localhost:~/mydir# cat myfile
line1
localhost:~/mydir#
```

echo "line2" >> myfile

```
localhost:~/mydir# echo "line2" >> myfile
localhost:~/mydir#
```

cat myfile **# Display the contents of myfile**

```
localhost:~/mydir# cat myfile
line1
line2
localhost:~/mydir#
```

cd .. *# move back to root directory. Note the space after cd before ..*

```
localhost:~/mydir# cd ..
localhost:~#
```

Pwd *# print working directory*

```
localhost:~# pwd
/root
localhost:~#
```

cd mydir *# move to mydir directory*

```
localhost:~# cd mydir
localhost:~/mydir#
```

cp myfile myfile2 *# copy file into a new file*

```
localhost:~/mydir# cp myfile myfile2
localhost:~/mydir#
```

cat myfile2 *# Display contents of the newly created myfile2*

```
localhost:~/mydir# cat myfile2
line1
line2
localhost:~/mydir#
```

ls -alrth myfile2 myfile *# list metadata on myfile & myfile2*

```
localhost:~/mydir# ls -alrth myfile2 myfile
-rw-r--r--    1 root     root          12 Sep 15 16:35 myfile
-rw-r--r--    1 root     root          12 Sep 15 16:39 myfile2
localhost:~/mydir#
```

rm -i myfile2 *# type y to confirm deletion and hit enter*

```
localhost:~/mydir# rm -i myfile2
rm: remove 'myfile2'? y
localhost:~/mydir#
```

cd .. *# move back to root directory*

```
localhost:~/mydir# cd ..
localhost:~#
```

rmdir mydir *# won't work because there's a file in there*

```
localhost:~# rmdir mydir
rmdir: 'mydir': Directory not empty
localhost:~#
```

rm -rf mydir *# VERY dangerous command, use with caution*

```
localhost:~# rm -rf mydir
localhost:~#
```

ls ***# ls is list files in a directory. Not you will see that myfile and myfile2 are deleted as is mydir directory***

```
localhost:~# ls
bench.py    hello.c     hello.js    readme.txt
localhost:~#
```

This should give you an intuitive understanding of how to navigate between directories (cd), print the current working directory (pwd), print the contents of files (cat), list the contents of directories (ls), copy files (cp), rename files (mv), list files (ls), move files (mv again), and remove files and directories (rm).

2.2.2 Text viewers and editors

In the next part, you will create a file. If you wish to view or edit files in Linux, there are a number of quick methods. This page is for users unfamiliar with text editors on the Linux platform. *Feel free to skip if you wish.*

Nano Text Editor

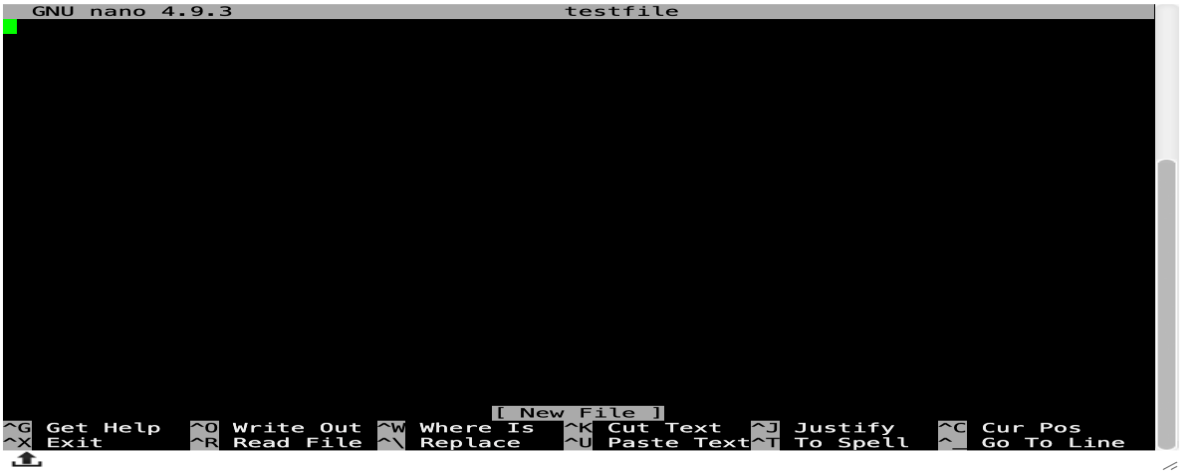
Nano is possibly the simplest way to edit a file for the rest of this course but we look at vi below as well. To create a new file.

in the terminal type**:**

nano testfile

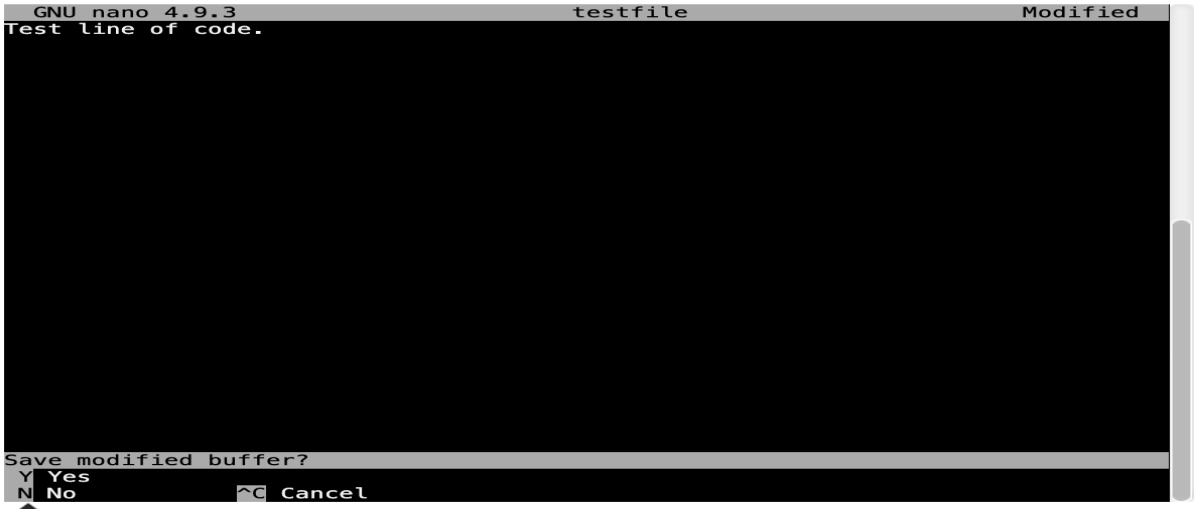
```
localhost:~# nano testfile
```

This will then open the nano text editor. Here you can enter any sample line of test you wish as shown next.



Enter the following line:

Test line of code.



To save and exit, type **CTRL + X.** You then type **Y** to confirm saving and exiting.

If you do a list command **ls**, you will see the file has been created.

ls *# list the files in current directory.*

File Viewer – cat

cat is a simple little program that displays the contents of a text file when you give the file name as an argument to it:

```
localhost:~# cat testfile
```

You will see the file contents displayed.

This is a nice way of viewing short files that fit on your screen, but if the file is so long that its contents cannot be displayed on your screen all at once, you will end up only staring at the end of the file. Maybe not exactly what you want. In most cases, you will want to use less instead.

File Viewer – less

less is a program that lets you view text files, like cat does, but if the files are so long that they don't fit on your screen, less automatically paginates the file. You use less by giving the file name as an argument to it:

```
localhost:~# less testfile
```

This allows you to more easily control scrolling through large files.

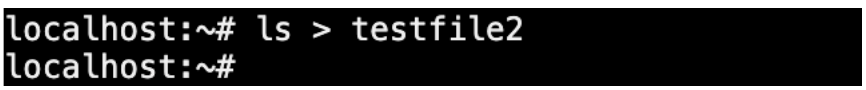
When viewing the file, you can use Page Up and Page Down keys to move through the file.

Typing **q** will exit.

You can also open several files at the same time so you can navigate from one file to next without closing it first.

To test this, we will create a second file. Let us do that the lazy way by 'piping' the output of the 'list files' command to a file called testfile 2 by typing the following:

ls > testfile2



So, if you want to open several files, just give all the file names at once:

less testfile testfile2.

You will then see the following.

To see the testfile2 contents, type **:n** *(note that is a colon followed by n)*

To move back to the testfile contents, type ***:p*** *(This stands for the previous file.)*

To quit type :q

:q

Text Editor – vi

Nano is a simple text editor, but hard-core Linux users adore vi. vi is generally considered the de facto standard in Unix editors because:

- It is usually available on all the flavours of Unix system.
- Its implementations are very similar across the board.
- It requires very few resources and is more user-friendly than other editors such as the ed or the ex.

You can use the vi editor to edit an existing file or to create a new file from scratch. You can also use this editor to just read a text file. *An improved version of the vi editor which is called the VIM is what we actually use.*

While working with the vi editor, we usually come across the following two modes −

Command mode − This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.

Insert mode − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file.

vi always starts in the command mode. To enter text, you must be in the insert mode for which simply type i. To come out of the insert mode, press the Esc key, which will take you back to the command mode. Hint − If you are not sure which mode you are in, press the Esc key twice; this will take you to the command mode. You open a file using the vi editor. Start by typing some characters and then come to the command mode to understand the difference.

To Start vi

To use vi on a file, type:

vi testfile

The first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text. It should appear as follows:

To Exit vi

Usually the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file.

Note: The cursor moves to bottom of screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key.

:x<Return> *quit vi, writing out modified file to file named in original invocation*

:wq<Return> *quit vi, writing out modified file to file named in original invocation*

:q<Return> *quit (or exit) vi*

:q!<Return> *quit vi even though latest changes have not been saved for this vi call*