School of Computing and Information Systems

COMP30023: Computer Systems

Lab Week 2

1 Introduction

In this workshop you will (re)?gain[1] familiarity with connecting to remote Linux instances through SSH, the Linux command line, the gcc and Makefiles. We will also focus on how processor internals affect program runtimes.
If you are unfamiliar with Linux/Un*x commands, please start by reading the tutorials listed in the Resources section.

2 Connecting to remote server using SSH

Throughout this workshop we will be executing commands on a server at a remote location. The remote server will be the instance created for you on the Melbourne Research Cloud (`https://dashboard.cloud.unimelb.edu.au/project`).

The Melbourne Research Cloud offers free on-demand computing resources to researchers at the University of Melbourne (and affiliated institutions). It provides similar functionality to commercial cloud providers such as Amazon Web Services, Microsoft Azure and Google Cloud Platform.

We will be connecting to your Virtual Machine (VM) instance using the SSH protocol. We will use your private key (created in the prelab) to authenticate with the server. You will also need the IP address of your instance which was sent to you.

Let's get connected!

2.1 *nix based system

If you are using a *nix based system (such as MacOS, some flavour of Linux), you can use the following command to directly log in to the remote instance.

```
ssh -i <path_to_key> <university_username>@<Instance_IP_address_>
```

Please note that though we use "**ubuntu**" as our user name in the example below, you will need to use your own university user name to login.
Note also that your key must not be readable by anyone except the owner. This can be achieved by the command

```
chmod 600 <path_to_key>
```

[1]Regex all the things!

If you are using WSL (Windows Subsystem for Linux), then the private key should be under your home directory, not a directory shared with Windows, because Windows' file system does not support Unix permissions.

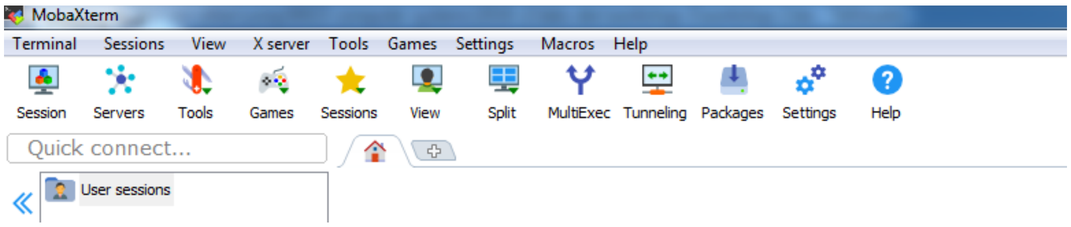
2.2 Windows

We will be using the MobaXterm (`https://mobaxterm.mobatek.net/`) software[2] on Windows to connect to the server.

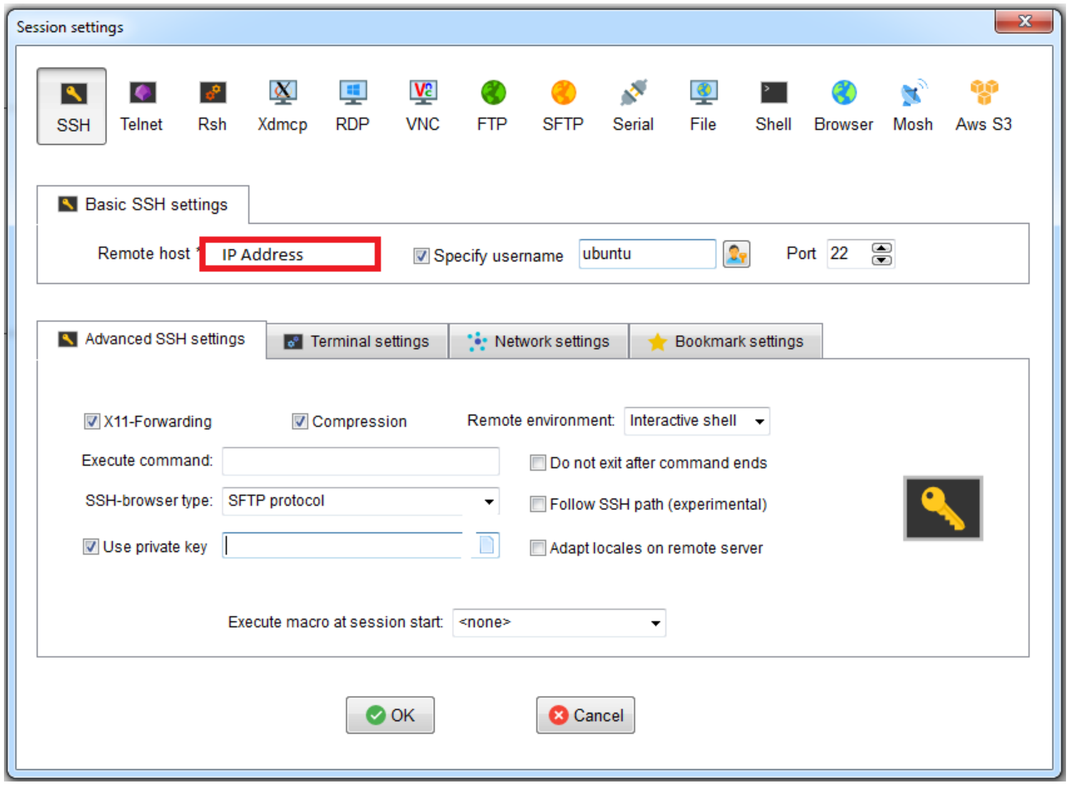
1. If you are using your personal computer, download and install MobaXterm Home Edition, which is free. (`https://mobaxterm.mobatek.net/`)

   MobaXterm is preinstalled on laboratory computers.

2. Click on the `Session` button


MobaXterm
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
User sessions

3. Select the `SSH` connection type.
   - Use the **IP address** of your instance in the 'Remote host' field
   - Select the checkbox to 'Specify username' and fill in as `<university_username>`.
   - Fill in **'22'** for Port.

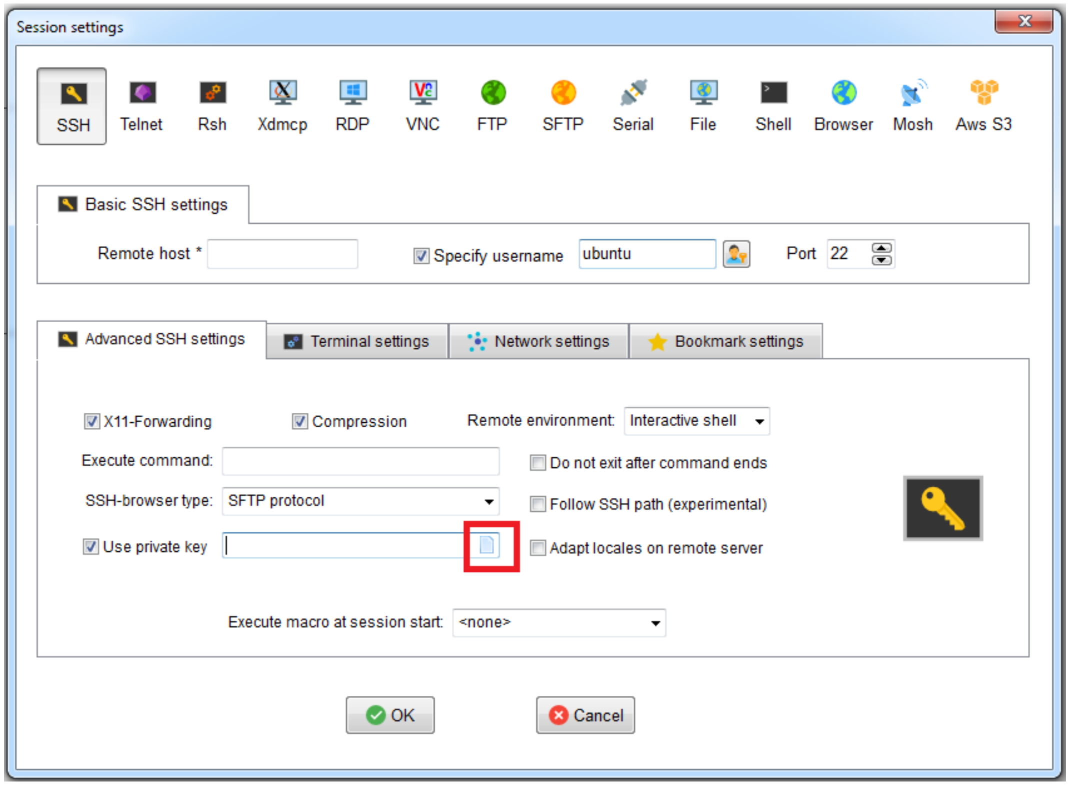


Remote host * IP Address
☑ Specify username ubuntu
Port 22

☑ X11-Forwarding ☑ Compression Remote environment: Interactive shell
Execute command:
☐ Do not exit after command ends
SSH-browser type: SFTP protocol
☐ Follow SSH path (experimental)
☑ Use private key
☐ Adapt locales on remote server
Execute macro at session start: <none>

[2]You may choose to use a different software. Some alternatives are PuTTY, mRemoteNG and Bitvise

4. Select your private key file
    - Click on the ‘Advanced SSH Settings’ tab
    - Select the ‘Use private key’ checkbox
    - Click on the small file-shaped icon to the right of the textbox
    - Browse to the location of your private key file and select it

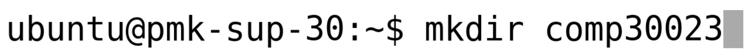


Remote host *
[✓] Specify username: ubuntu
Port: 22

[✓] X11-Forwarding
[✓] Compression
Remote environment: Interactive shell
Execute command:
[ ] Do not exit after command ends
SSH-browser type: SFTP protocol
[ ] Follow SSH path (experimental)
[✓] Use private key
[ ] Adapt locales on remote server
Execute macro at session start: <none>

3 Creating files on server

1. Let’s create a sub-directory inside our home (`~`) directory.
    - Look up the online manual (man) on mkdir:
      Command: `man mkdir`
      You can look up the online manual on any Linux command, and should do that before asking for help.
    - Create the directory: Command: `mkdir comp30023`

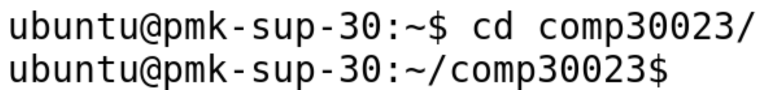


```
ubuntu@pmk-sup-30:~$ mkdir comp30023
```

Remark. In the above screenshot, “ubuntu” is the user name and “pmk - sup - 30” is the name of the machine. You might see different text in your own machine.

2. List the contents of the directory to confirm sub-directory creation.
   Command: `ls`

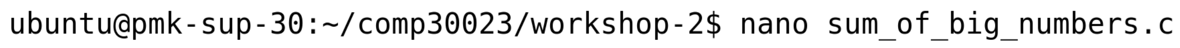
3. Navigate to the created sub-directory



```
ubuntu@pmk-sup-30:~$ cd comp30023/
ubuntu@pmk-sup-30:~/comp30023$
```

4. Use the commands you learnt above to create a subdirectory within the `comp30023` directory called `workshop-2`, and then navigate to the newly created `workshop-2` folder.

5. Open a new file named `sum_of_big_numbers.c` using the `nano`[3] text editor.[4]

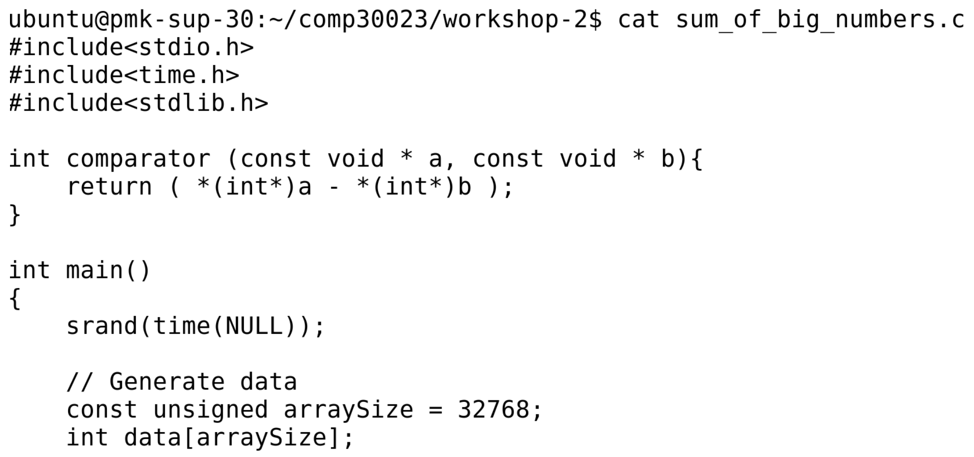


```
ubuntu@pmk-sup-30:~/comp30023/workshop-2$ nano sum_of_big_numbers.c
```

6. Copy and paste the code in the listing `sum_of_big_numbers.c` provided in "Modules/Sample Code" on the LMS (Hint: you can use the keyboard shortcut *Shift + Insert* to paste from the clipboard).

7. Save the file (*Ctrl + O*) and exit the nano editor (*Ctrl + X)*.

8. Confirm that the file has been written properly using the `cat` command (Note: Only the top portion of the output has been shown here to save space.)



```
ubuntu@pmk-sup-30:~/comp30023/workshop-2$ cat sum_of_big_numbers.c
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

int comparator (const void * a, const void * b){
    return ( *(int*)a - *(int*)b );
}

int main()
{
    srand(time(NULL));

    // Generate data
    const unsigned arraySize = 32768;
    int data[arraySize];
```

9. You can also show the contents of a file using the commands `more` and `less`. These show the file one screenful at a time, and allow you to search to contents. Look them up in the manual. When running `more` or `less`, you can type "h" to get a list of internal commands. The output of `man` uses `less`, and so it is worth learning how to navigate in it. Play around. Even if you are familiar with *nix, try to find one new feature of `less` that you didn't know.

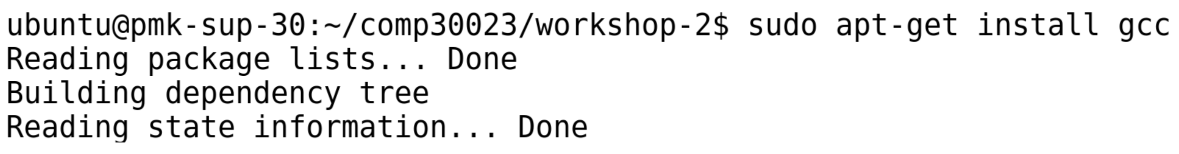
[3]Once you are comfortable with Linux, you are strongly recommended to ride out the steep learning curve of a more powerful text editor such as `vi`. You could also settle for emacs.
`https://en.wikipedia.org/wiki/Editor_war`

[4]Do not simply transfer the .c files on to the server. It is a useful skill to be comfortable with editing files directly on a remote server

4 Compiling files

1. We will be using the `gcc` compiler to compile our C file. Let's install it using the command `apt-get`[5]

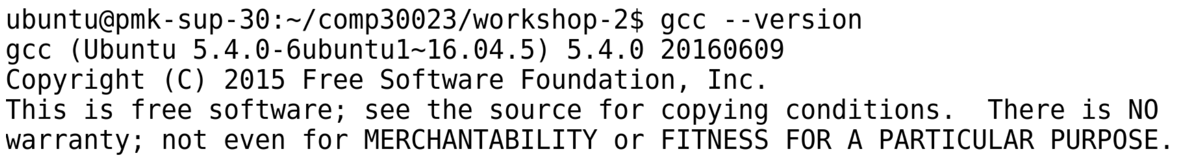
   Command: `man apt-get` Command: `$ apt-get install gcc`[6]

   ```
   ubuntu@pmk-sup-30:~/comp30023/workshop-2$ sudo apt-get install gcc
   Reading package lists... Done
   Building dependency tree
   Reading state information... Done
   ```

   (Again, only part of the output is shown)

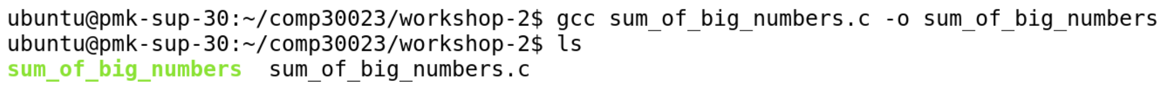
2. Confirm correct installation of the `gcc` compiler (your version may differ) Command: `$ gcc --version`

   ```
   ubuntu@pmk-sup-30:~/comp30023/workshop-2$ gcc --version
   gcc (Ubuntu 5.4.0-6ubuntu1~16.04.5) 5.4.0 20160609
   Copyright (C) 2015 Free Software Foundation, Inc.
   This is free software; see the source for copying conditions.  There is NO
   warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
   ```

3. Compile the C file. We use the `-o` switch to `gcc` to name the executable file `sum_of_big_numbers`

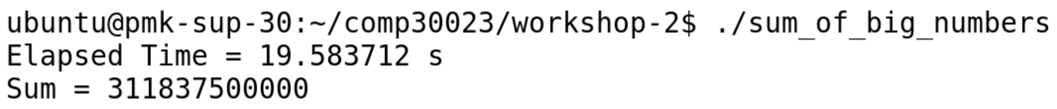
   Command: `$ gcc sum_of_big_numbers.c -o sum_of_big_numbers`[7]

   ```
   ubuntu@pmk-sup-30:~/comp30023/workshop-2$ gcc sum_of_big_numbers.c -o sum_of_big_numbers
   ubuntu@pmk-sup-30:~/comp30023/workshop-2$ ls
   sum_of_big_numbers  sum_of_big_numbers.c
   ```

4. Run the file.

   Command: `$ ./sum_of_big_numbers`

   ```
   ubuntu@pmk-sup-30:~/comp30023/workshop-2$ ./sum_of_big_numbers
   Elapsed Time = 19.583712 s
   Sum = 311837500000
   ```

   The output is the sum of the numbers greater than or equal to 128 in the array, and the time taken to run.

5. Uncomment the line that sorts the array (using qsort), recompile, and run the program to notice an interesting result in relation to the execution time.

   Explanation at the end of this sheet!

[5]The Advanced Package Tool (APT) provides easy software management on Debian-based Linux distributions (such as Ubuntu).

[6]You may need to run `apt-get update` first

[7]You may need to add the -std=c99 flag

5 Projects with multiple files, Compiler Flags

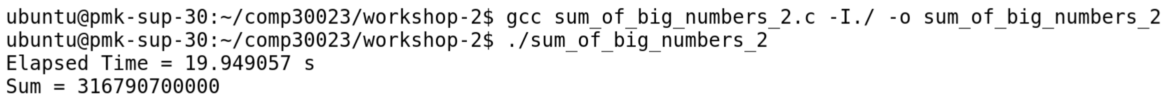
It is good programming practice to separate out your code to separate files based on functionality. Let's separate out the sorting functionality in our program.

1. Create the files `sum_of_big_numbers_2.c` and `sort.h` from the listings provided on the LMS. [8]

2. Note the use of Include Guards (`#ifndef` related preprocessor directives) in `sort.h`. Use of Include Guards is good programming practice, prevents double includes, and will potentially save you a lot of grief in your projects.

3. Compile the files to one executable file. Notice that we have to provide the location of the header file using the `-I` switch to `gcc` as the current directory (`./`)

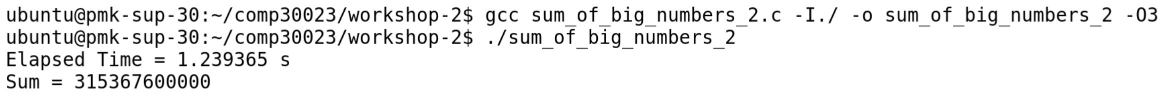
   Command: `$ gcc sum_of_big_numbers_2.c -I./ -o sum_of_big_numbers_2`

```
ubuntu@pmk-sup-30:~/comp30023/workshop-2$ gcc sum_of_big_numbers_2.c -I./ -o sum_of_big_numbers_2
ubuntu@pmk-sup-30:~/comp30023/workshop-2$ ./sum_of_big_numbers_2
Elapsed Time = 19.949057 s
Sum = 316790700000
```

4. `gcc` provides a number of options to control the compilation process. One such option is the optimisation level. Let's set the optimisation to Level 3 (using switch -O3) and observe the running time.

   Command: `$ gcc sum_of_big_numbers_2.c -I./ -o sum_of_big_numbers -O3`

```
ubuntu@pmk-sup-30:~/comp30023/workshop-2$ gcc sum_of_big_numbers_2.c -I./ -o sum_of_big_numbers_2 -O3
ubuntu@pmk-sup-30:~/comp30023/workshop-2$ ./sum_of_big_numbers_2
Elapsed Time = 1.239365 s
Sum = 315367600000
```

   We see a large decrease in running time. However, note that this is a contrived example - real-world gains may not be as drastic.

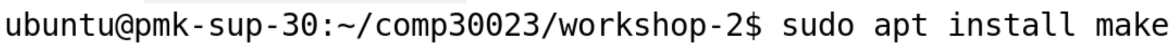
6 Makefiles

We saw that even for the simple example above that the command to run `gcc` got quite long and unwieldy.

The `make` program uses configuration files called 'Makefiles' to streamline the process of compiling large projects with multiple files and options.

1. Let's first install the `make` program

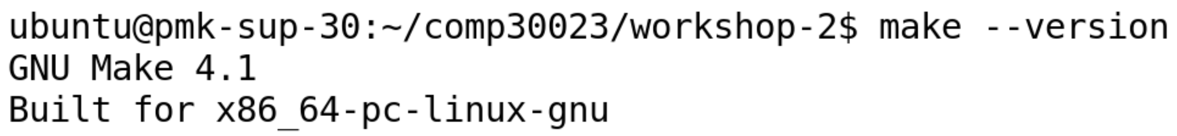
   Command: `$ apt-get install make`

```
ubuntu@pmk-sup-30:~/comp30023/workshop-2$ sudo apt install make
```

   Command: `$ make --version`

[8] You could investigate how to transfer the files downloaded to your computer to the remote machine using `scp` or a GUI-based FTP client such as WinSCP



```
ubuntu@pmk-sup-30:~/comp30023/workshop-2$ make --version
GNU Make 4.1
Built for x86_64-pc-linux-gnu
```

2. Create a file named `Makefile` using the listing given on the LMS. This is the configuration file that the `make` program will use.

   Use the `make` command to compile

   ```
   ubuntu@pmk-sup-30:~/comp30023/workshop-2$ make
   gcc -c -o sum_of_big_numbers_2.o sum_of_big_numbers_2.c -Wall -Wextra -std=gnu99 -I.
   gcc -o exec_big_sum sum_of_big_numbers_2.o -Wall -Wextra -std=gnu99 -I.
   ```

3. Run `sum_of_big_numbers_2`

   Command: `$ ./sum_of_big_numbers_2`

4. Experiment with `make clean` (delete object files) and `make clobber` (delete both object and executable files).

5. Add the `gcc` compiler flag `-O3` to the Makefile and see if you can generate an executable that runs fast as we saw before.

   You can use this Makefile as the basis for your projects.

7 Bonus: Explanation of Runtime Results

We observed that it was faster to process a sorted array compared to an unsorted array. What could be the reason for this? We have to go through all the elements in both cases so this cannot be related to computational complexity.

The key lies in a circuit called the *branch predictor* in the CPU. The branch predictor tries to predict whether a given condition will evaluate to true or false based on the evaluation history of the condition. The CPU will fetch the next instructions in the guessed branch and *speculatively execute* them to improve concurrency. If the guess was wrong, the results of the speculative execution are discarded.

With a sorted array, we would have a sequence of values for which the condition will be false, followed by a sequence of values for which the condition will be true. This behaviour is easy to predict, compared to the unsorted case where you will be constantly rolling back to the branch which takes time. The `O3` optimisation level includes an optimisation which allows the compiler to generate what is called a *conditional move* (among other optimisations), which avoids the incorrect branch prediction penalty.

7.1 Speculative Execution in the News

Major security flaws that affected modern microprocessors was frontpage news in media outlets around the world in early January 2018[9]. Named *Meltdown* and *Spectre*, both flaws exploited the

[9] `https://www.nytimes.com/2018/01/03/business/computer-flaws.html`
`http://www.abc.net.au/news/science/2018-01-04/intel-chip-flaw-a-security-threat/9303280`

aggressive speculative execution performed in Intel microprocessors.

Kernel memory is protected and meant to only be accessible to the Operating System (and not user programs.) *Meltdown* exploited the fact that Intel microprocessors allow user programs to speculatively use kernel data, where the access check happens some time after the instruction starts executing. Though the speculative execution is properly blocked, the impact of the speculation on processor internals (especially the processor cache) can be used to infer kernel memory values. This allows potentially any kernel memory to be accessed by user programs.

Spectre is a more general attack involving speculative execution features on array bounds checks and branching instructions. This attack also potentially allows the leaking of information from kernel memory to user programs, and could potentially affect AMD and ARM systems in addition to Intel processors.

8 Resources

8.1 Learning Linux

- `https://linuxjourney.com/lesson/the-shell`
- `http://www.ee.surrey.ac.uk/Teaching/Unix/unix1.html`

8.2 Bonus Material on Branch Prediction, Speculative Execution and Processor Flaws

- `https://stackoverflow.com/questions/11227809/why-is-it-faster-to-process-a-sorted-array-than-an-unsorted-array/11227902#11227902`
- `https://en.wikipedia.org/wiki/Branch_predictor`
- `https://en.wikipedia.org/wiki/Speculative_execution`
- `https://arstechnica.com/gadgets/2018/01/meltdown-and-spectre-every-modern-processor-has-unfixable-security-flaws/`
- `https://arstechnica.com/gadgets/2018/01/whats-behind-the-intel-design-flaw-forcing-numerous-patches/`