Name: Archit Benipal

Subject: Software Supply Chain Security (CY 653)

Assignment 9: Other BCC tools

Biolatency BCC tool:

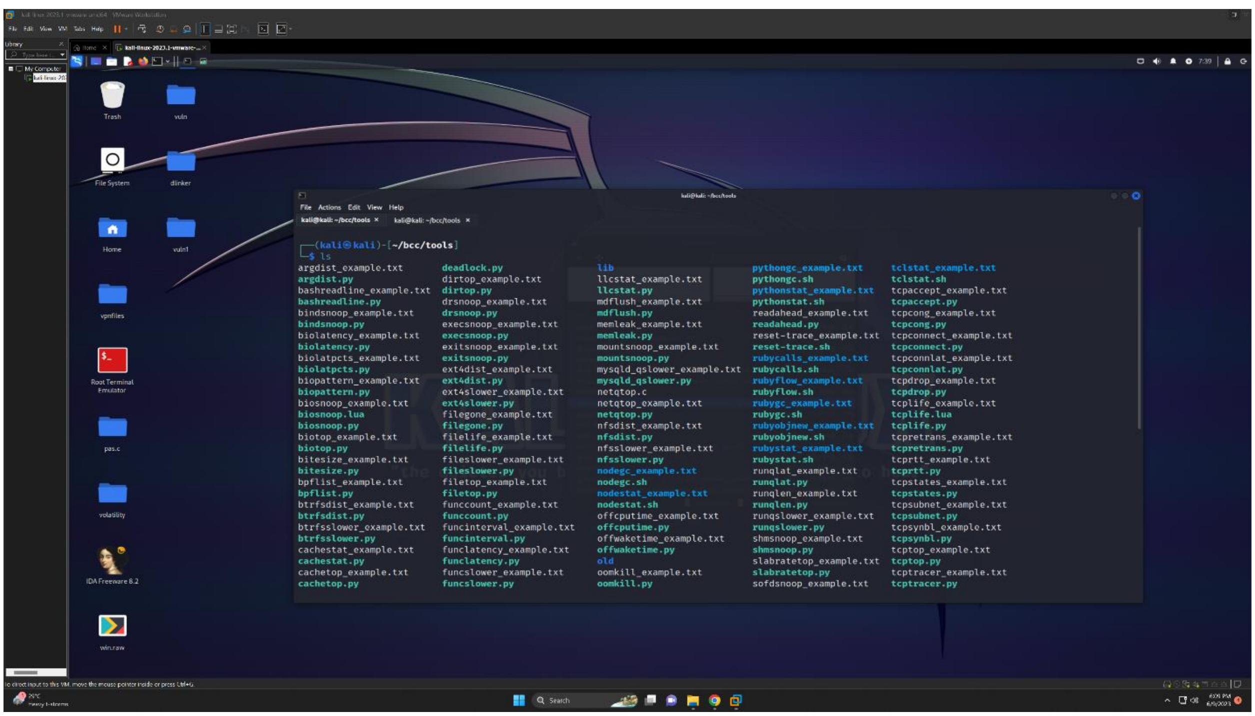
What is biolatency:

It is a tool included in the BCC (BPF Compiler Collection) that uses eBPF (Extended Berkeley Packet Filter) to track the completion time of block device I/O (input/output) operations, often referred to as block I/O or simply bio. It effectively measures the latency of disk I/O operations in a Linux system.

When it is executed, it creates a histogram showing the distribution of I/O latency in milliseconds. This can be useful for identifying whether your storage subsystem is experiencing delays. It might help in diagnosing storage performance issues and can be part of a comprehensive performance analysis methodology.

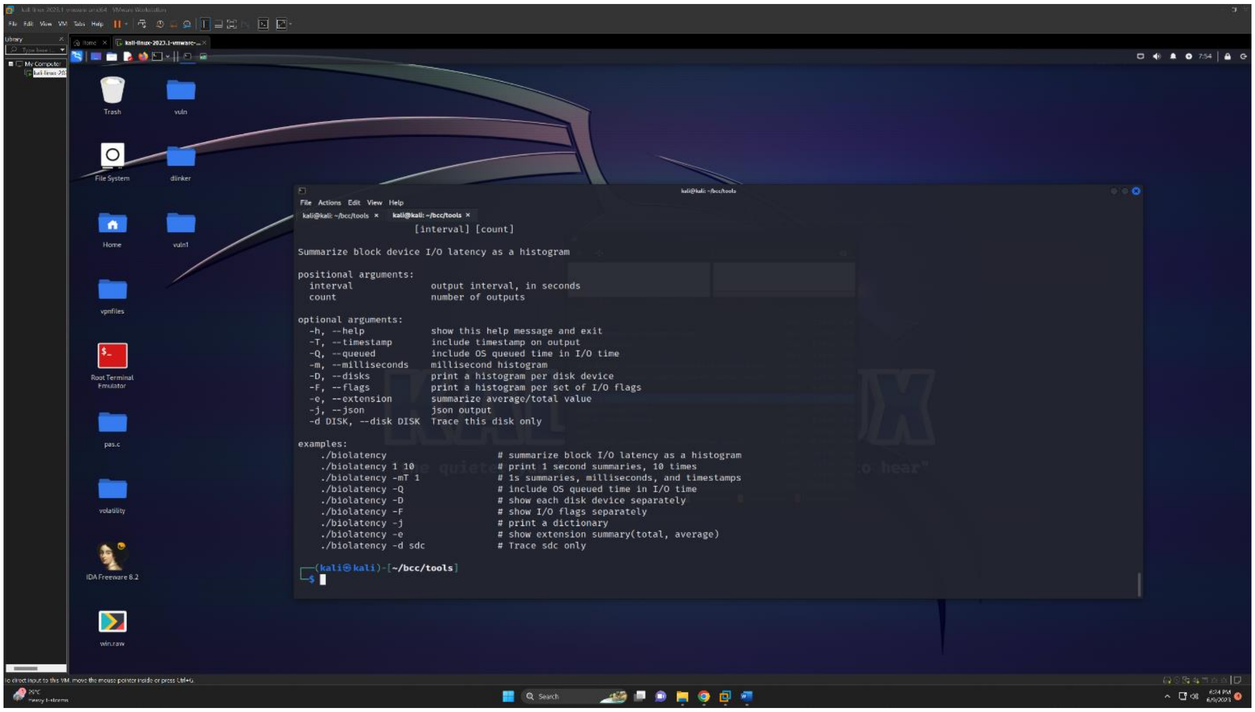
Usage:

Step 1: Cloned the Github repo of the bcc tools and navigated to the /bcc/tools directory to list all the tools



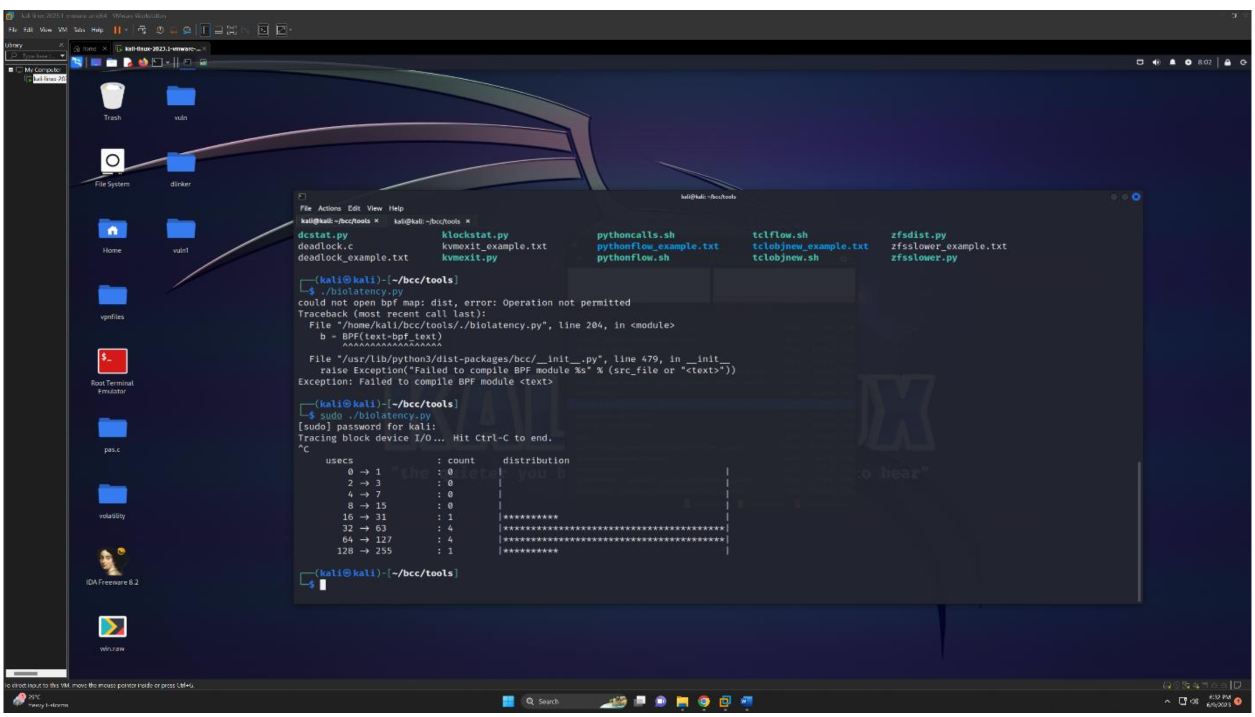
Here we can see list of several tools with the instruction manual for the usage of each tool and access to the old tools available in the previous versions of BCC.

Step 2: Examples



Here we can see all the examples with this script.

Step 3: Executed the biolatency script with no switches



Breakdown:

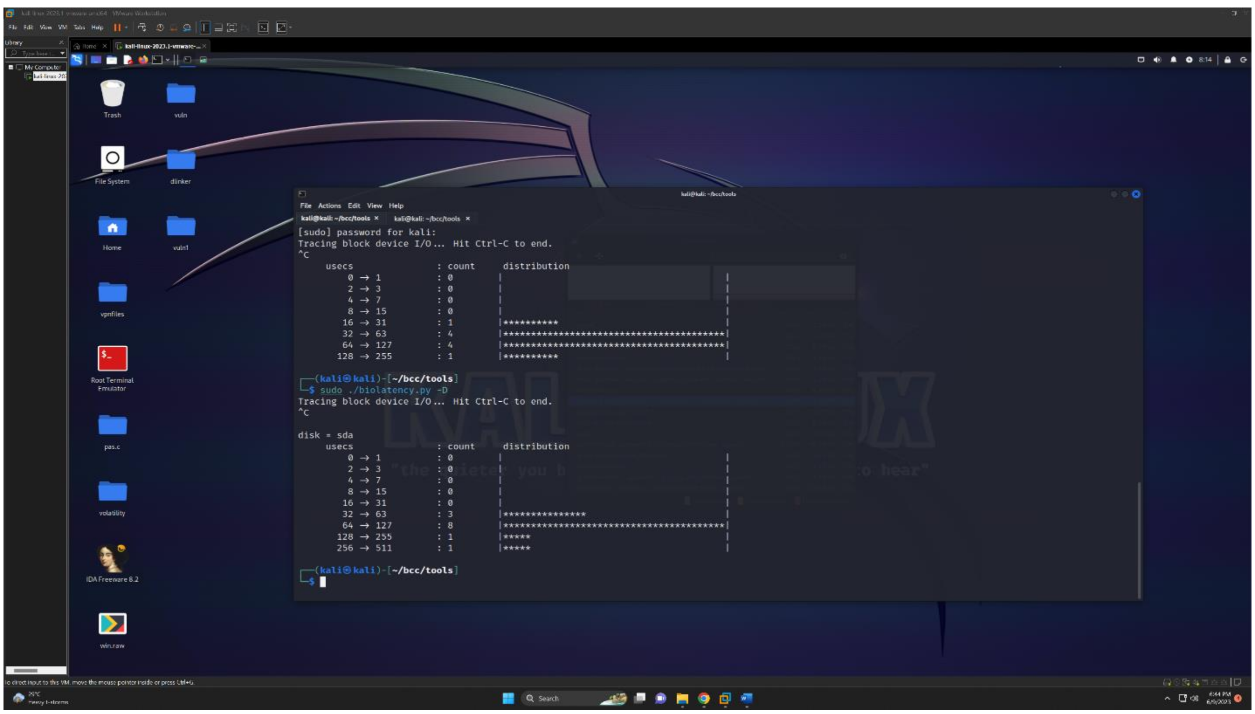
The script traces block device I/O latency

- The script attaches to the block device I/O tracing mechanism and starts collecting data on the latency of I/O operations.
- The output displays a histogram-like representation of the I/O latency distribution.
- The column labeled "usecs" represents the different ranges of I/O latency in microseconds.
- The column labeled "count" shows the number of I/O operations that fall within each latency range.
- The bar graph represents the distribution of I/O operations across the latency ranges, with the "*" character used to fill the bars.
- The longer the bar, the higher the count of I/O operations within that particular latency range.

Here it can also be observed that

- There were no I/O operations with a latency in the range of 0 to 7 microseconds.
- There was one I/O operation with a latency in the range of 16 to 31 microseconds.
- There were four I/O operations each with latencies in the ranges of 32 to 63 microseconds and 64 to 127 microseconds.
- There was one I/O operation with a latency in the range of 128 to 255 microseconds.

Step 4: Ran the script with the -D switch to show each disk separately



Breakdown:

As we have allotted only one disk to the VM we can see only one disk in the output

Here it can be Observed that:

- There were no I/O operations with a latency in the range of 0 to 15 microseconds.
- There were three I/O operations with latencies in the range of 32 to 63 microseconds.
- There were eight I/O operations with latencies in the range of 64 to 127 microseconds.
- There was one I/O operation with a latency in the range of 128 to 255 microseconds.
- There was one I/O operation with a latency in the range of 256 to 511 microseconds.

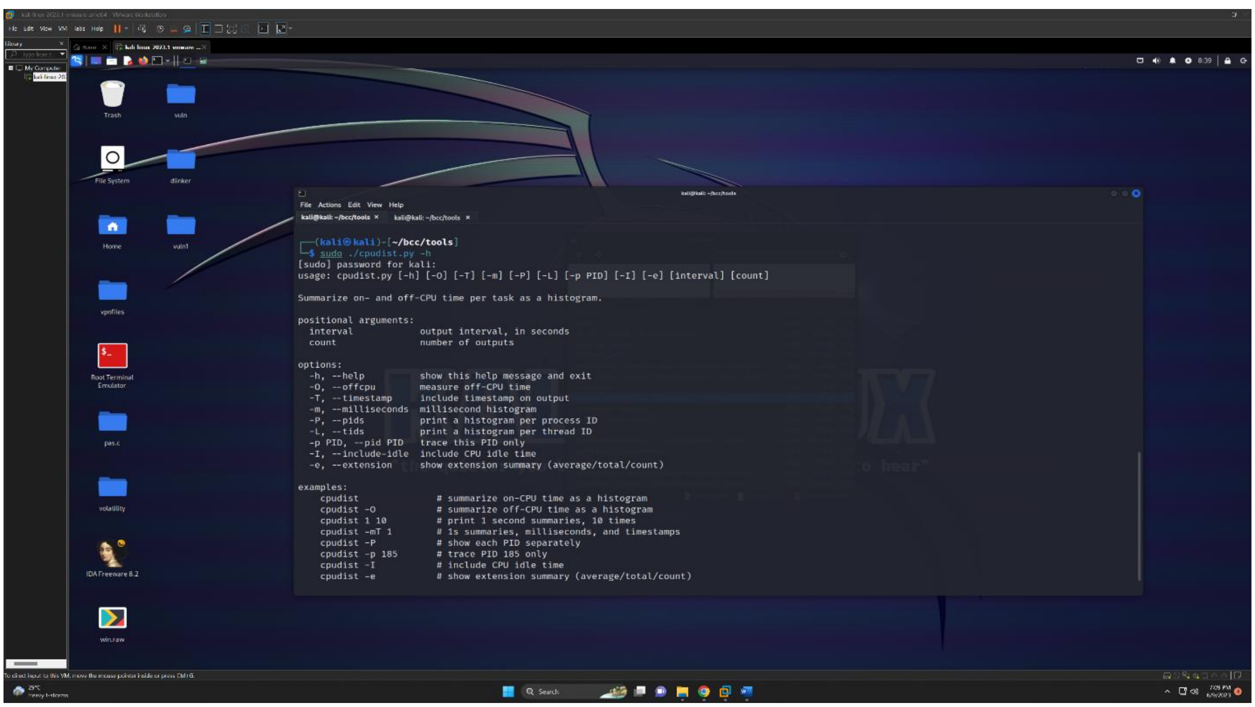
CPUdist BCC tool:

What is CPUdist

It is a BCC-based tool that uses eBPF (Extended Berkeley Packet Filter) to measure CPU distribution in a Linux system. More specifically, **cpudist** measures the distribution of CPU time spent by threads while they are running on a CPU. This includes only the time spent running on a CPU, not while waiting in run queues. It creates a histogram showing the distribution of time spent on CPU. This information can be useful for identifying whether there are any performance issues related to CPU usage. The output might assist in diagnosing CPU-bound applications or identifying CPU performance anomalies.

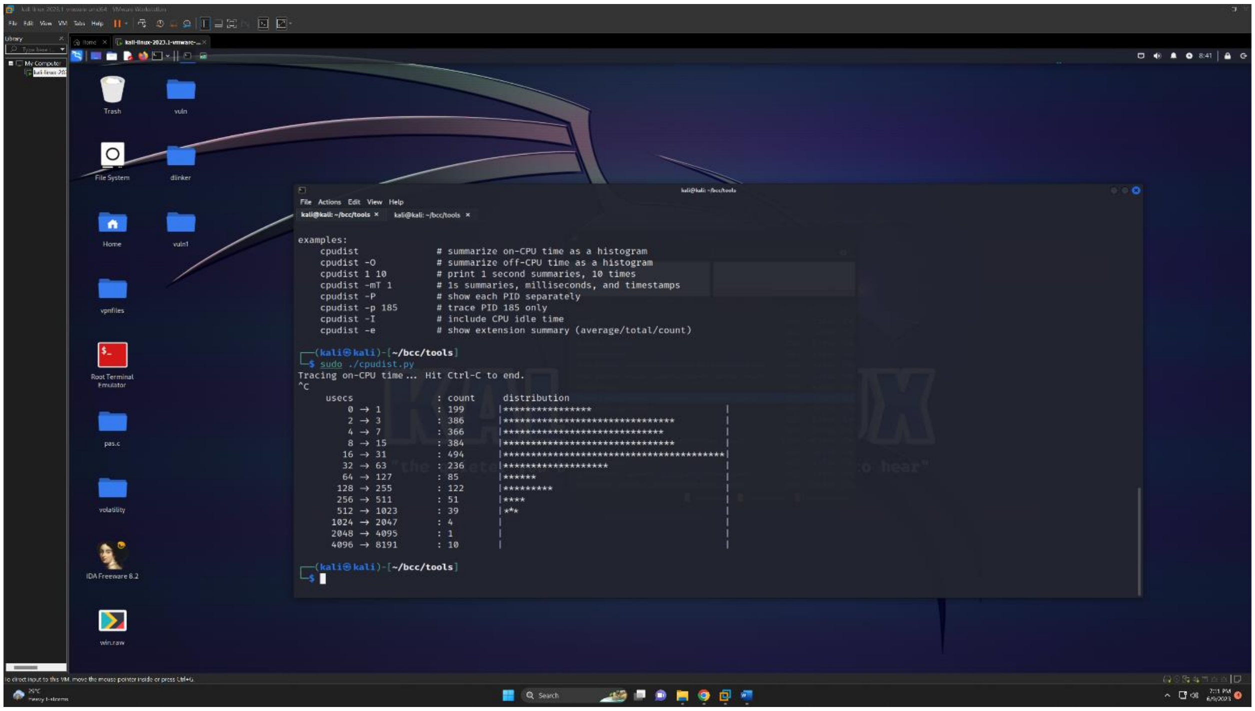
Usage:

Step 1: Ran the –help command on cpudist.py



Here we can see the basic examples and the usage of cpudist.py script

Step 2: Executing the cpudist.py



Breakdown:

This script traces the on-CPU time, which refers to the amount of time that a process or thread spends executing on the CPU.

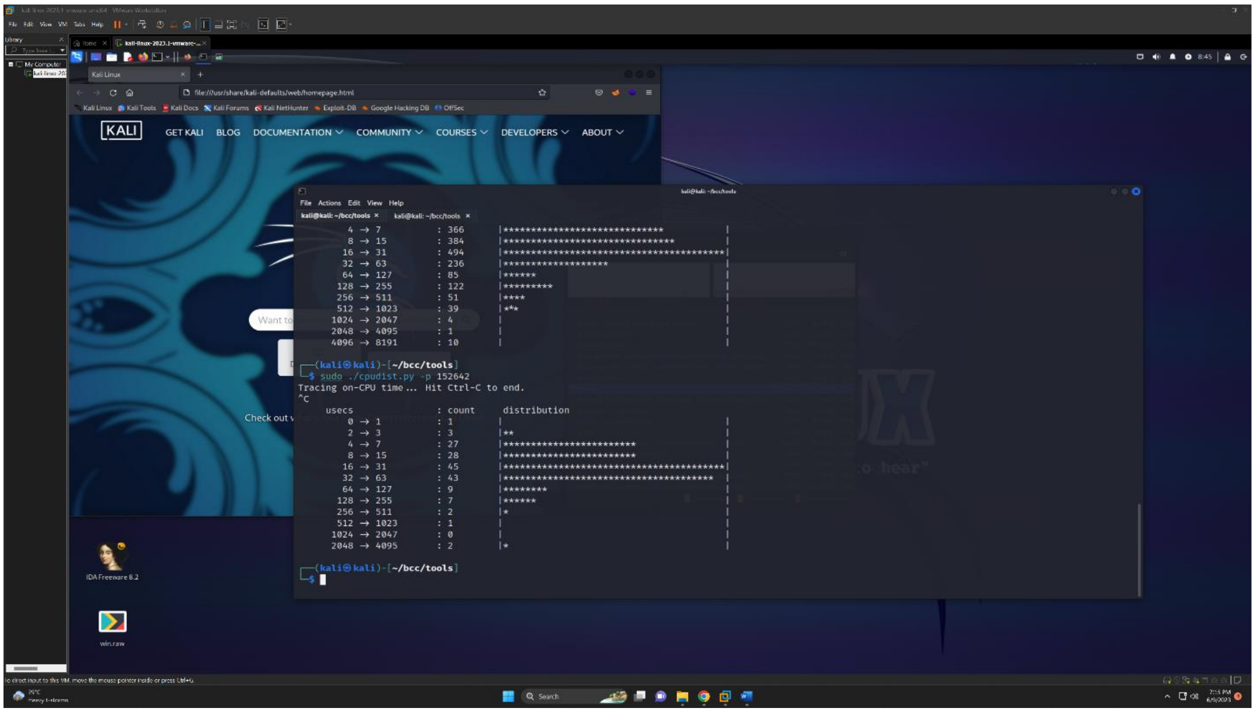
- The column labeled "usecs" represents the different ranges of on-CPU time in microseconds.
- The column labeled "count" shows the number of events (in this case, the number of times a process or thread had a particular on-CPU time) that fall within each time range.
- The bar graph represents the distribution of events across the time ranges, with the "*" character used to fill the bars.

It can also be observed that:

- There were 199 events with on-CPU time in the range of 0 to 1 microsecond.
- There were 386 events with on-CPU time in the range of 2 to 3 microseconds.
- There were 366 events with on-CPU time in the range of 4 to 7 microseconds.
- There were 384 events with on-CPU time in the range of 8 to 15 microseconds.
- There were 494 events with on-CPU time in the range of 16 to 31 microseconds.
- There were 236 events with on-CPU time in the range of 32 to 63 microseconds.
- There were 85 events with on-CPU time in the range of 64 to 127 microseconds.
- There were 122 events with on-CPU time in the range of 128 to 255 microseconds.
- There were 51 events with on-CPU time in the range of 256 to 511 microseconds.
- There were 39 events with on-CPU time in the range of 512 to 1023 microseconds.
- There were 4 events with on-CPU time in the range of 1024 to 2047 microseconds.

- ➢ There was 1 event with on-CPU time in the range of 2048 to 4095 microseconds.
- ➢ There were 10 events with on-CPU time in the range of 4096 to 8191 microseconds.

Step 3: Tracing a particular PID

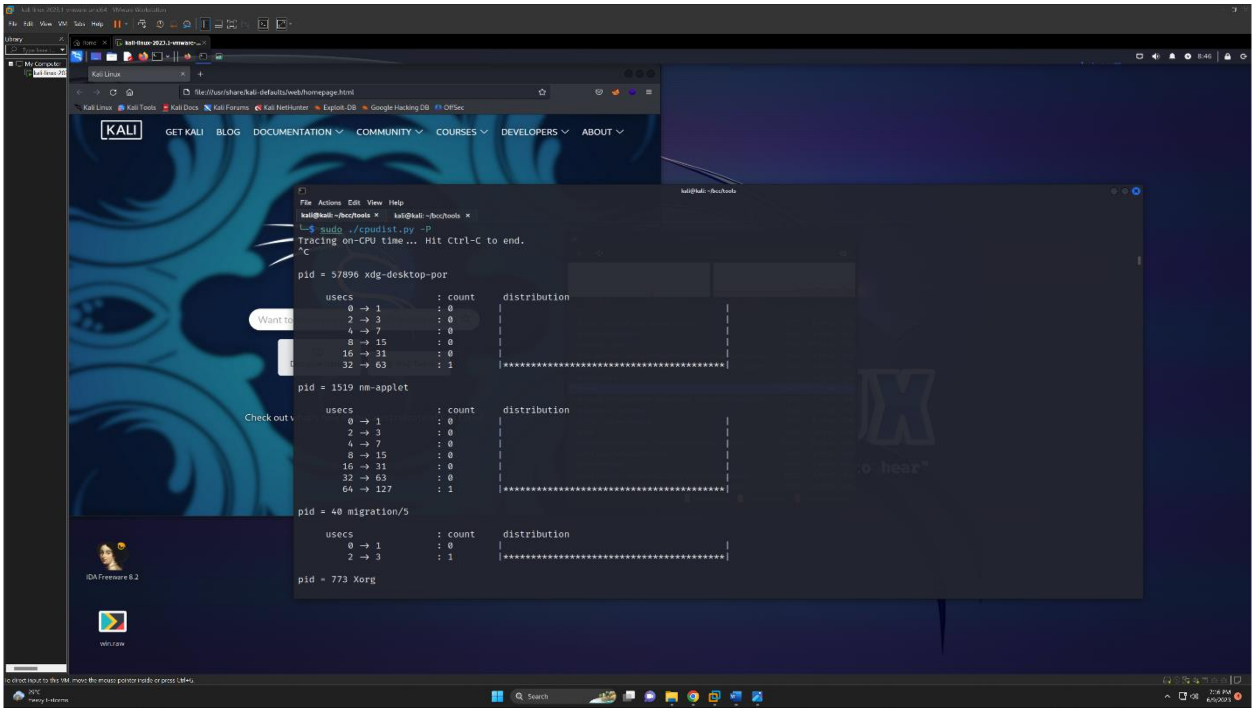


Breakdown:

Here we launched firefox and tried to trace the PID , It can also be observed that

- ➢ There was 1 event with on-CPU time in the range of 0 to 1 microsecond.
- ➢ There were 3 events with on-CPU time in the range of 2 to 3 microseconds.
- ➢ There were 27 events with on-CPU time in the range of 4 to 7 microseconds.
- ➢ There were 28 events with on-CPU time in the range of 8 to 15 microseconds.
- ➢ There were 45 events with on-CPU time in the range of 16 to 31 microseconds.
- ➢ There were 43 events with on-CPU time in the range of 32 to 63 microseconds.
- ➢ There were 9 events with on-CPU time in the range of 64 to 127 microseconds.
- ➢ There were 7 events with on-CPU time in the range of 128 to 255 microseconds.
- ➢ There were 2 events with on-CPU time in the range of 256 to 511 microseconds.
- ➢ There was 1 event with on-CPU time in the range of 512 to 1023 microseconds.
- ➢ There were 0 events with on-CPU time in the range of 1024 to 2047 microseconds.
- ➢ There were 2 events with on-CPU time in the range of 2048 to 4095 microseconds.

Step 4: View each PID separately



Command used “sudo ./cpudist.py -P”

Breakdown:

For process with PID 57896 (**xdg-desktop-por**):

- There were no events with on-CPU time in the range of 0 to 63 microseconds.
- There was 1 event with on-CPU time in the range of 32 to 63 microseconds.

For process with PID 1519 (**nm-applet**):

- There were no events with on-CPU time in the range of 0 to 63 microseconds.
- There was 1 event with on-CPU time in the range of 64 to 127 microseconds.

Team Name:

- ALL Safe

Members:

- Archit Benipal
- Siva Prasad Kolli
- Venkata Nethaji Yenduri