Name: Archit Benipal

Subject: Software Supply Chain Security (CY 653)

Assignment 8: BPFTRACE

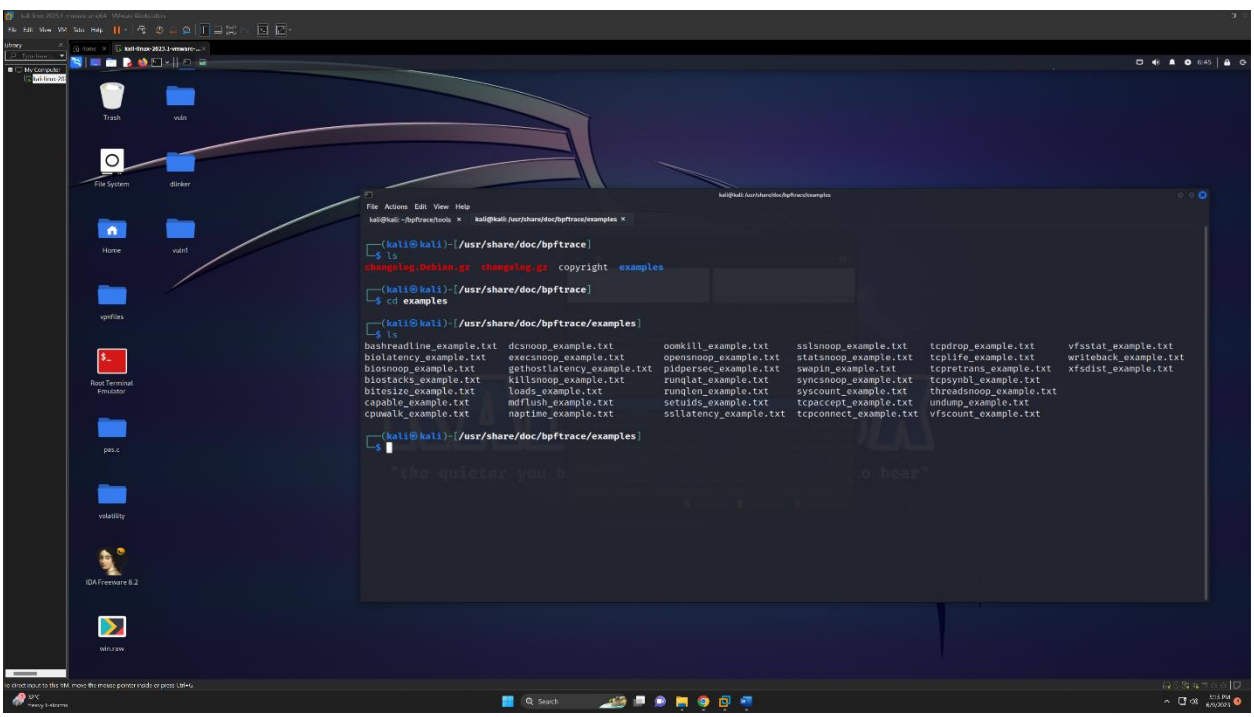
What is 'bpftrace'

It is a high-level tracing language for Unix-based operating systems that provides a dynamic tracing capability. It uses Berkeley Packet Filter (BPF) syntax, and its functionality is built on top of the eBPF (Extended Berkeley Packet Filter) functionality in the Linux kernel. It is a very powerful tool that allows you to write scripts that can trace system events, user events, tracepoints, and more. It's commonly used for performance analysis, debugging, monitoring, and understanding underlying system behavior.

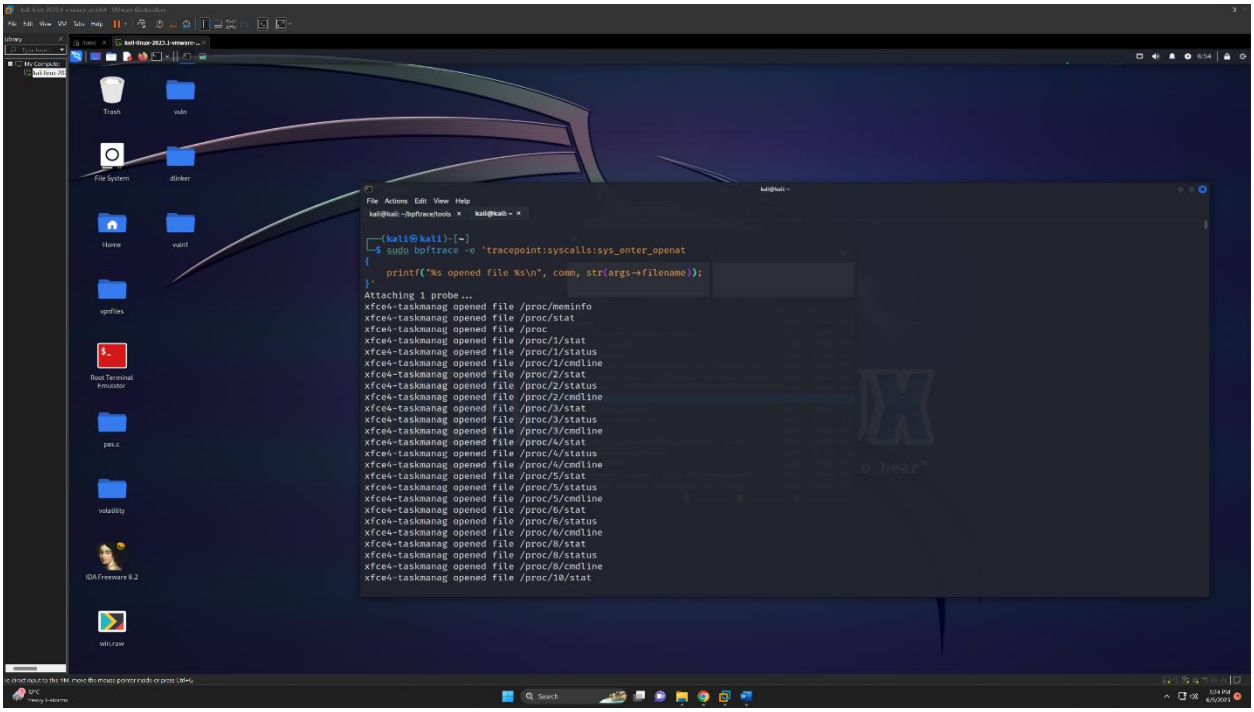
All being said it is a flexible tool and its scripts can become quite complex, allowing you to trace and monitor a wide range of system behaviors. However, because it uses eBPF to interact with the kernel, running **bpftrace** scripts typically requires root privileges.

Usage:

Step 1: Installed the bpftrace using command "sudo apt-get -y bpftrace", but did not get the tools in the /doc folder



Step 2: Ran some examples found online

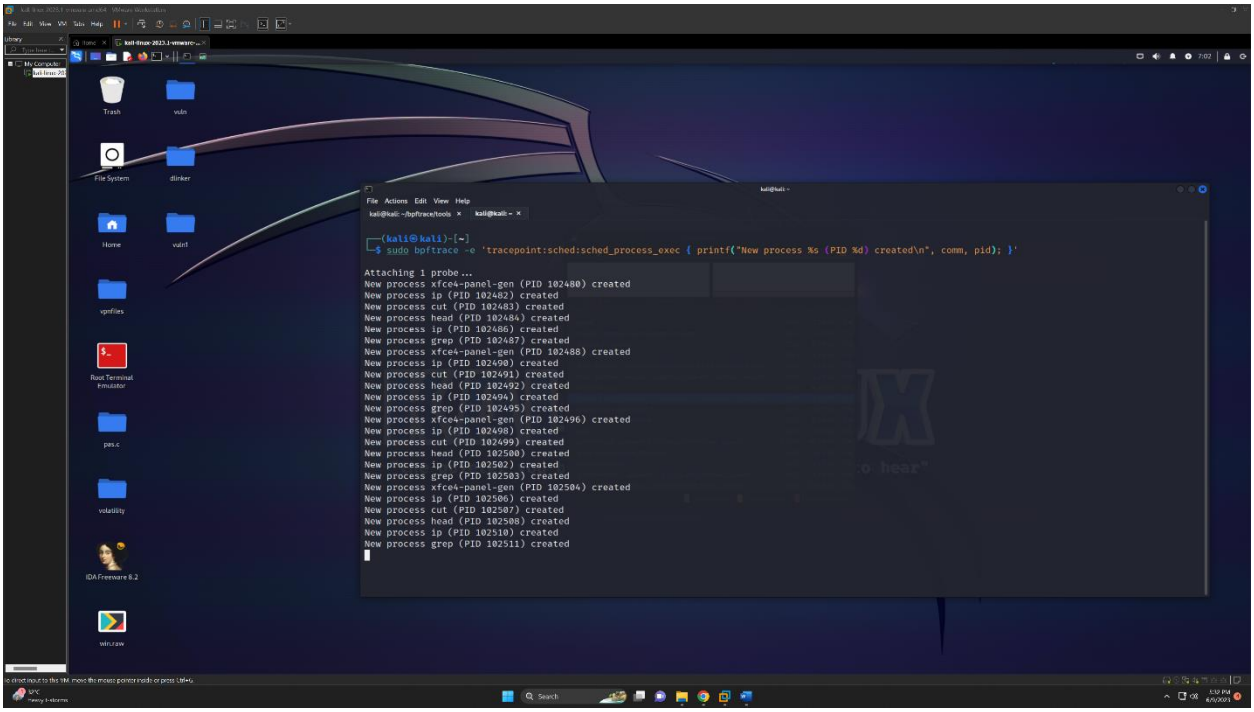


Breakdown:

This script uses a tracepoint to monitor calls to the **openat** system call, and prints a message each time a file is opened. It uses **comm** to print the name of the process that opened the file, and **str(args->filename)** to print the name of the file.

- **xfce4-taskmanag opened file /proc/meminfo**: The process **xfce4-taskmanag** opened the file **/proc/meminfo**. This file contains information about the system's memory usage.
- **xfce4-taskmanag opened file /proc/stat**: The process **xfce4-taskmanag** opened the file **/proc/stat**. This file provides various system statistics, such as CPU usage, process counts, and more.
- **xfce4-taskmanag opened file /proc/1/status**: The process **xfce4-taskmanag** opened the file **/proc/1/status**. This file provides various status information about the process with PID 1.
- **xfce4-taskmanag opened file /proc/1/cmdline**: The process **xfce4-taskmanag** opened the file **/proc/1/cmdline**. This file contains the command-line arguments passed to the process with PID 1.

Step 3: Ran another online script that can show new processes being created



Breakdown:

This script captures the process name (**comm**) and process ID (**pid**) whenever a new process is created.

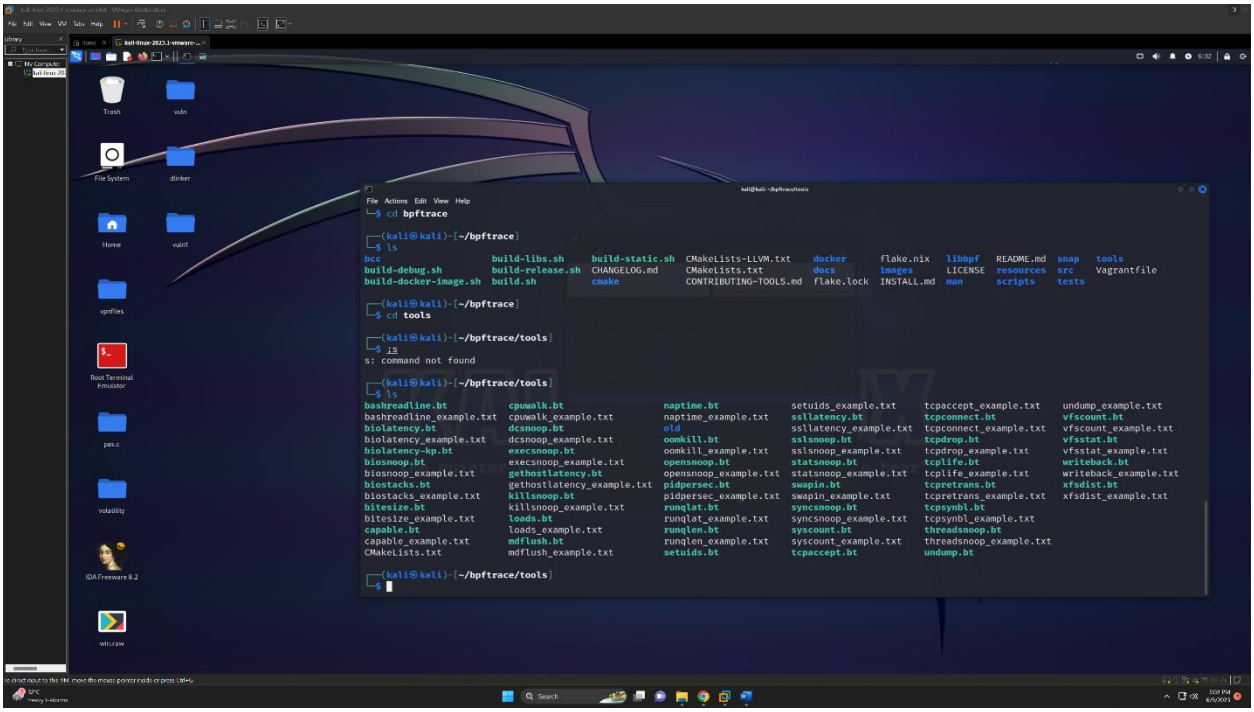
In the output we can see several new processes created but we have discussed the top 4 processes:

- **New process xfce4-panel-gen (PID 102480) created**: A new process with the name **xfce4-panel-gen** and process ID 102480 was created.
- **New process ip (PID 102482) created**: A new process with the name **ip** and process ID 102482 was created. The **ip** command is typically used for configuring network interfaces and routing tables.
- **New process cut (PID 102483) created**: A new process with the name **cut** and process ID 102483 was created. The **cut** command is commonly used for manipulating and extracting columns of text.
- **New process head (PID 102484) created**: A new process with the name **head** and process ID 102484 was created. The **head** command is often used to display the beginning portion of files or input streams.

the output shows the '**sched_process_exec'** tracepoint being triggered whenever a new process is created. The script captures the process name and process ID and prints a message indicating the creation of the new process.

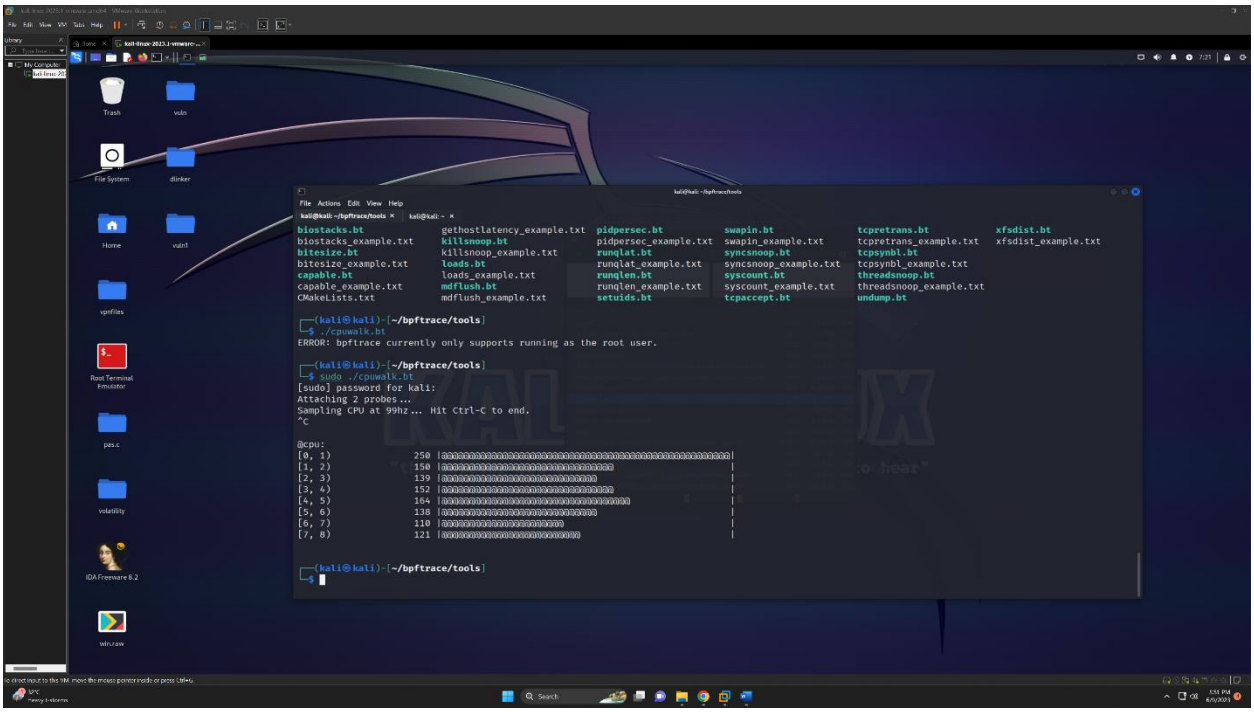
'sched_process_exec' is a tracepoint in the Linux kernel that is triggered when a new process is created. It allows us to trace and capture information about process execution events.

Step 4: We cloned the git repo for the bpftrace and accessed the /tools directory.



```
└─$ cd bpftrace

┌──(kali㉿kali)-[~/bpftrace]
└─$ ls
bcc                    build-libs.sh     build-static.sh  CMakeLists-LLVM.txt    docker      flake.nix   libbpf   README.md  snap   tools
build-debug.sh         build-release.sh  CHANGELOG.md     CMakeLists.txt         docs        images      LICENSE  resources  src    Vagrantfile
build-docker-image.sh  build.sh          cmake            CONTRIBUTING-TOOLS.md  flake.lock  INSTALL.md  man      scripts    tests

┌──(kali㉿kali)-[~/bpftrace]
└─$ cd tools

┌──(kali㉿kali)-[~/bpftrace/tools]
└─$ is
s: command not found

┌──(kali㉿kali)-[~/bpftrace/tools]
└─$ ls
bashreadline.bt           cpuwalk.bt                  naptime.bt             setuids_example.txt     tcpaccept_example.txt    undump_example.txt
bashreadline_example.txt  cpuwalk_example.txt         naptime_example.txt    ssllatency.bt           tcpconnect.bt            vfscount.bt
biolatency.bt             dcsnoop.bt                  old                    ssllatency_example.txt  tcpconnect_example.txt   vfscount_example.txt
biolatency_example.txt    dcsnoop_example.txt         oomkill.bt             sslsnoop.bt             tcpdrop.bt               vfsstat.bt
biolatency-kp.bt          execsnoop.bt                oomkill_example.txt    sslsnoop_example.txt    tcpdrop_example.txt      vfsstat_example.txt
biosnoop.bt               execsnoop_example.txt       opensnoop.bt           statsnoop.bt            tcplife.bt               writeback.bt
biosnoop_example.txt      gethostlatency.bt           opensnoop_example.txt  statsnoop_example.txt   tcplife_example.txt      writeback_example.txt
biostacks.bt              gethostlatency_example.txt  pidpersec.bt           swapin.bt               tcpretrans.bt            xfsdist.bt
biostacks_example.txt     killsnoop.bt                pidpersec_example.txt  swapin_example.txt      tcpretrans_example.txt   xfsdist_example.txt
bitesize.bt               killsnoop_example.txt       runqlat.bt             syncsnoop.bt            tcpsynbl.bt
bitesize_example.txt      loads.bt                    runqlat_example.txt    syncsnoop_example.txt   tcpsynbl_example.txt
capable.bt                loads_example.txt           runqlen.bt             syscount.bt             threadsnoop.bt
capable_example.txt       mdflush.bt                  runqlen_example.txt    syscount_example.txt    threadsnoop_example.txt
CMakeLists.txt            mdflush_example.txt         setuids.bt             tcpaccept.bt            undump.bt

┌──(kali㉿kali)-[~/bpftrace/tools]
└─$ 
```

Step 5: Executed the cpuwalk.bt script under /bpftrace/tools



```
biostacks.bt              gethostlatency_example.txt  pidpersec.bt           swapin.bt               tcpretrans.bt            xfsdist.bt
biostacks_example.txt     killsnoop.bt                pidpersec_example.txt  swapin_example.txt      tcpretrans_example.txt   xfsdist_example.txt
bitesize.bt               killsnoop_example.txt       runqlat.bt             syncsnoop.bt            tcpsynbl.bt
bitesize_example.txt      loads.bt                    runqlat_example.txt    syncsnoop_example.txt   tcpsynbl_example.txt
capable.bt                loads_example.txt           runqlen.bt             syscount.bt             threadsnoop.bt
capable_example.txt       mdflush.bt                  runqlen_example.txt    syscount_example.txt    threadsnoop_example.txt
CMakeLists.txt            mdflush_example.txt         setuids.bt             tcpaccept.bt            undump.bt

┌──(kali㉿kali)-[~/bpftrace/tools]
└─$ ./cpuwalk.bt
ERROR: bpftrace currently only supports running as the root user.

┌──(kali㉿kali)-[~/bpftrace/tools]
└─$ sudo ./cpuwalk.bt
[sudo] password for kali:
Attaching 2 probes...
Sampling CPU at 99hz... Hit Ctrl-C to end.
^C

@cpu:
[0, 1)               250 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[1, 2)               150 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                     |
[2, 3)               139 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                        |
[3, 4)               152 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                     |
[4, 5)               164 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                  |
[5, 6)               138 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                        |
[6, 7)               110 |@@@@@@@@@@@@@@@@@@@@@@                              |
[7, 8)               121 |@@@@@@@@@@@@@@@@@@@@@@@@@                           |


┌──(kali㉿kali)-[~/bpftrace/tools]
└─$ 
```

Breakdown:

As we have allotted only 8 processors in the VM we can see the workload of each processes

In this case, we can see CPU(0) taking most load. If only one CPU was taking the work load then that would have been a matter for investigation.

- The script attaches two probes to gather information about the CPU usage.
- The output shows a histogram-like representation of CPU utilization over different time intervals (from 0 to 8, each interval representing 1/99th of a second).
- The numbers on the left side represent the count of samples falling within each interval.
- The bar graph represents the relative magnitude of CPU usage within each interval, with the "@" character used to fill the bars.
- The longer the bar, the higher the CPU usage during that particular time interval.

References:

- https://opensource.com/article/19/8/introduction-bpftrace
- https://github.com/iovisor/bpftrace

Team Name:

- ALL Safe

Members:

- Archit Benipal
- Siva Prasad Kolli
- Venkata Nethaji Yenduri