

Name: Archit Benipal

Subject: Software Supply Chain Security (CY 653)

Assignment 7: Try another BCC example

What is BCC?

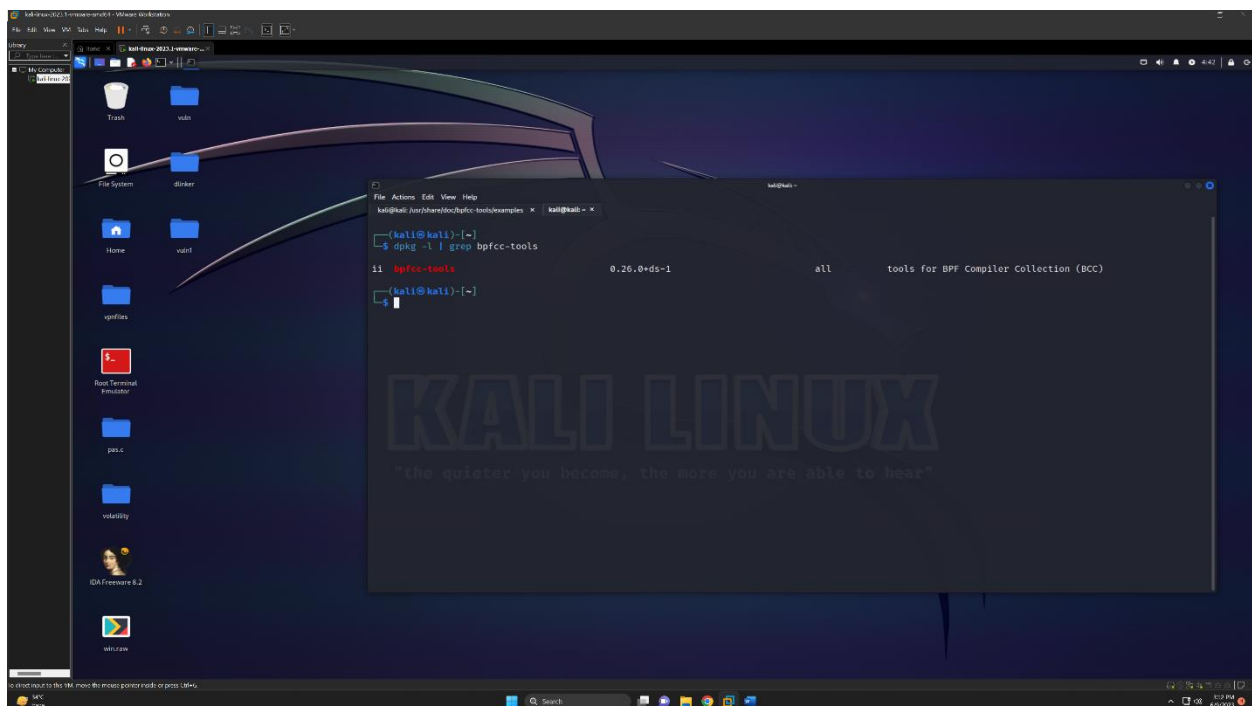
BCC stands for BPF Compiler Collection. It is a toolkit for creating efficient kernel tracing and manipulation programs, primarily using eBPF (extended Berkeley Packet Filter). BCC was developed by the IO Visor Project, which is an open-source project and a community of developers for creating IO and networking I/O functions.

It is important to note that BCC is meant for low-level system programming. You might use it for creating system performance tools, doing detailed systems introspection, or other advanced tasks. It is not generally used for everyday programming tasks.

BCC tools use Python as the front-end scripting language, with LLVM (Low-Level Virtual Machine) as a backend to compile BPF programs. The combination of Python and LLVM allows developers to write scripts that can perform a wide range of system monitoring tasks with high efficiency. As BCC evolves, it's important to refer to the latest documentation or its GitHub repository for the most current information.

Usage: Step 1:

Verified the installation of BCC



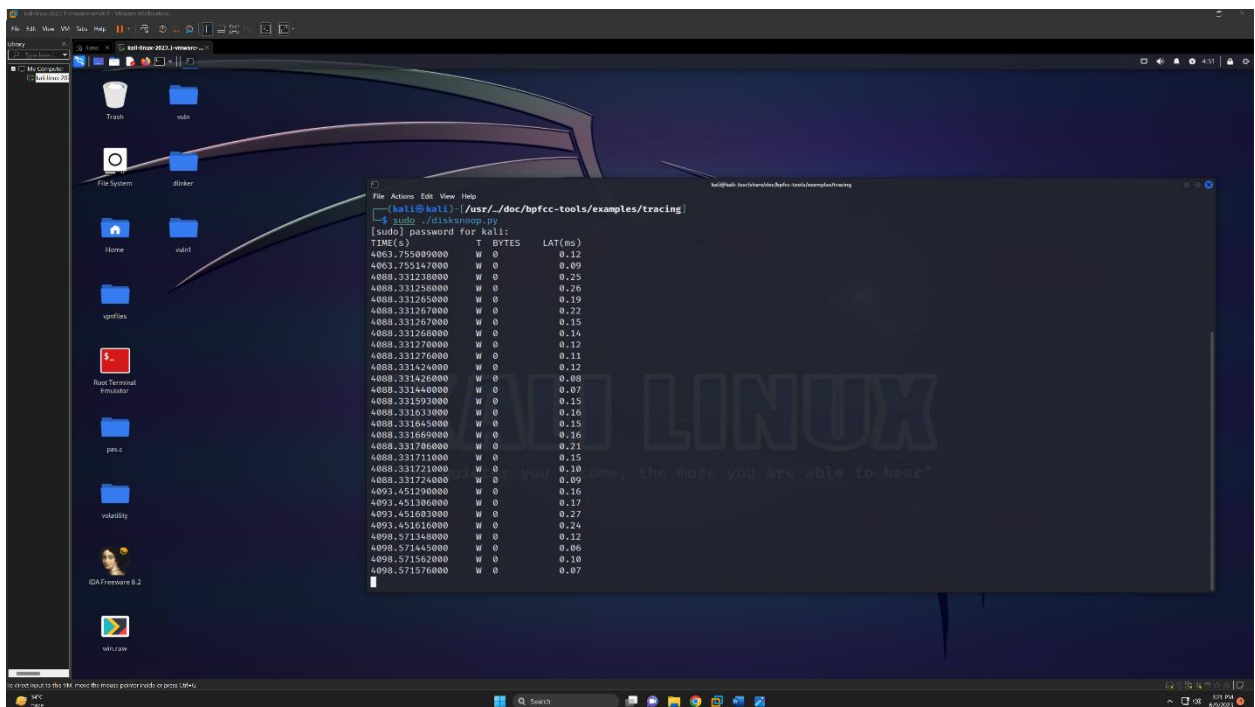
Breakdown of the command used: (dpkg -l | grep bpfcc-tools):

The **dpkg -l | grep bpfcc-tools** command is doing a couple of things.

- **dpkg -l**: This command lists all packages that are installed in system. **dpkg** is the package manager for Debian-based systems like Ubuntu, and the **-l** flag tells it to list packages.
- **|**: This is a pipe. It takes the output from the command on its left (in this case, **dpkg -l**) and uses it as the input to the command on its right.
- **grep bpfcc-tools**: This is a **grep** command, which searches the input it receives for lines containing a specified pattern. In this case, the pattern is **bpfcc-tools**, so **grep** will filter out all lines that don't contain **bpfcc-tools**.

Step 2:

Executed the **disksnoop.py** script under **/examples/tracing**



```
(kali@kali) ~/usr/local/bpfcc-tools/examples/tracing
$ sudo ./disksnoop.py
[sudo] password for kali:
TIME(s)      T  BYTES  LAT(ms)
4063.755090000  W  0      0.12
4063.755147000  W  0      0.09
4088.331230000  W  0      0.25
4088.331258000  W  0      0.26
4088.331258000  W  0      0.19
4088.331267000  W  0      0.22
4088.331267000  W  0      0.15
4088.331268000  W  0      0.14
4088.331270000  W  0      0.12
4088.331276000  W  0      0.11
4088.331424000  W  0      0.12
4088.331426000  W  0      0.08
4088.331440000  W  0      0.07
4088.331593000  W  0      0.15
4088.331633000  W  0      0.16
4088.331645000  W  0      0.15
4088.331669000  W  0      0.16
4088.331706000  W  0      0.21
4088.331711000  W  0      0.15
4088.331721000  W  0      0.10
4088.331724000  W  0      0.09
4093.451200000  W  0      0.16
4093.451306000  W  0      0.17
4093.451603000  W  0      0.27
4093.451616000  W  0      0.24
4098.571340000  W  0      0.12
4098.571445000  W  0      0.06
4098.571562000  W  0      0.18
4098.571576000  W  0      0.07
```

Breakdown:

What is **disksnoop**?

It is one of many tools provided by BPF compiler collection in **bpfcc-tools**. This particular script is used to trace block device I/O (input/output) and print statistics about it.

In simpler terms, **disksnoop.py** is a tool that helps to monitor and debug disk I/O operations on system.

This can be helpful for diagnosing performance issues or understanding the I/O behavior of specific processes or the system as a whole.

Block devices are a type of device can read from and write to, and they are called "block" devices because they handle data in blocks of a fixed size. Hard drives, SSDs, and USB drives are all examples of block devices. When **disksnoop.py** is run, it uses eBPF (Extended Berkeley Packet Filter) to hook into the kernel at the points where it handles block device I/O. It then collects information about these I/O operations, such as what process is performing the I/O, which block device the I/O is on, whether the I/O operation is a read or a write, and how much data is being read or written. This data can then be used to diagnose performance issues, understand how a particular piece of software uses disk I/O, or gather statistics about the system's overall I/O usage.

Note: This tool uses eBPF to hook into the kernel, it usually needs to be run with root permissions.

Breakdown of the command and output:

The output you're seeing from running **disksnoop.py** represents block device I/O (input/output) events in your system. Each line is a single I/O event.

- **TIME(s):** This is the timestamp of the I/O event, in seconds, from the start of the system (also known as system uptime). It helps to identify when exactly the I/O operation happened.
- **T:** This represents the type of I/O operation. 'R' stands for 'Read', and 'W' stands for 'Write'. In your case, all operations are 'W', meaning these are all write operations.
- **BYTES:** This column shows the size of the I/O operation in bytes. In your case, all entries show '0'.
- **LAT(ms):** This column represents the latency of the operation in milliseconds. Latency is the time it takes for an operation to complete. Lower latency usually means faster data processing.

Step 3: Executed undump.py with firefox as process selected

The screenshot displays a Kali Linux desktop environment. In the foreground, a terminal window is open, showing the execution of the `undump.py` script. The terminal output indicates that the script is tracing the process `firefox` (PID 60116). The output shows several lines of data, including timestamps, process IDs, and I/O statistics. A task manager window is also visible, showing a list of running processes, with `firefox` highlighted. The desktop background features a dark theme with various application icons and a search bar.

```
File Actions Edit View Help
kali@kali:~/disksnoop$ python3 undump.py --process=firefox
Tracing PID=60116 UNIX socket packet
PID 60116 Recv 12 bytes
00 00 0f 1a 11 c0 09 00 82 01 00
00 00 00 0f 02 bf 02 0f 02 9c

PID 60116 Recv 32 bytes
01 01 10 1a 00 00 00 00 83 86 a4
00 00 00 00 00 00 00 00 00 00 00

PID 60116 Recv 32 bytes
01 01 11 1a 00 00 00 00 83 86 a4
00 00 00 00 00 00 00 00 00 00 00

PID 60116 Recv 32 bytes
01 01 12 1a 00 00 00 00 83 86 a4
00 00 00 00 00 00 00 00 00 00 00

PID 60116 Recv 32 bytes
01 01 13 1a 00 00 00 00 83 86 a4
00 00 00 00 00 00 00 00 00 00 00

PID 60116 Recv 32 bytes
01 01 14 1a 00 00 00 00 83 86 a4
00 00 00 00 00 00 00 00 00 00 00

PID 60116 Recv 32 bytes
01 01 15 1a 00 00 00 00 83 86 a4
00 00 00 00 00 00 00 00 00 00 00
```

We can see here that as we start the firefox we start receiving packet in hexadecimal format.

Breakdown of the command and the output:

- In the command used we have executed the undump.py script with the process ID as 60116 which is the PID for firefox.
- **PID 60116 Recv 32 bytes / 44 bytes:** This is telling that the process with Process ID (PID) 60116 received a packet that was 32 bytes or 44 bytes in length.
- The hexadecimal lines are showing the data in the packet. The data is displayed in hexadecimal format, with each pair of digits representing a single byte.

Errors encountered while running BCC scripts after BCC installation:

[illegible]

Reason:

The Error indicates the BPF module failed to compile due to a lack of necessary kernel headers and the kheaders module. Current kernel seems to be **6.1.0-kali7-amd64**, and based on the error, it appears that the headers for this kernel version aren't installed on your system, or the directory **/lib/modules/6.1.0-kali7-amd64/build** is missing.

The 'kheaders' module is necessary for BCC to work properly. Some Linux distributions may not include this module depending on the kernel version.

Fix:

1. Updated the packages “sudo apt update && sudo apt upgrade”
2. Installed the kernel headers “sudo apt-get install linux-headers-\$(uname -r)”
3. Rebooted the system “sudo reboot”

By upgrading the system we also Updated the linux kernel version.

References used:

- <https://github.com/iovisor/bcc>

Team Name:

- ALL Safe

Members:

- Archit Benipal
- Siva Prasad Kolli
- Venkata Nethaji Yenduri