In HIS name

Ali Sheikh Attar

Nmap & ebpf security tools demo

- Nmap

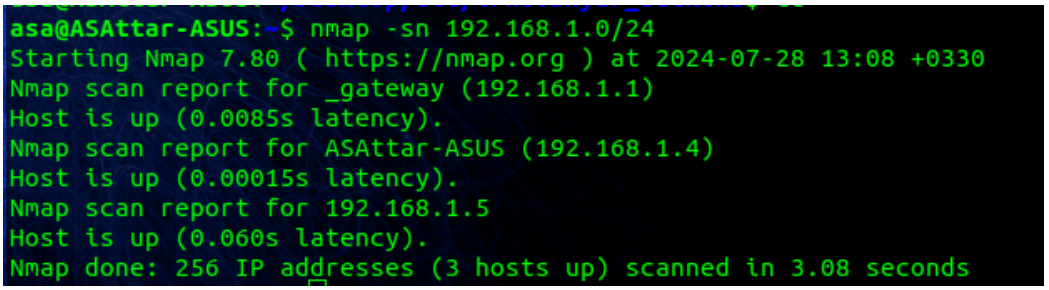
  For this demo i use my home router network

Network Inventory & Discovery

to identify all devices connected to a network

```
nmap -sn 192.168.1.0/24
```

This command performs a "ping scan" (-sn), which tells Nmap to only discover hosts without port scanning. It scans the entire subnet 192.168.1.0/24 to list all active devices.



output

Vulnerability Assessment

check if any devices on the network are vulnerable to a known exploit.

```
nmap --script vuln 192.168.1.0/24
```

Explanation: The --script vuln option tells Nmap to run a series of vulnerability detection scripts against the target network 192.168.1.0/24. This helps identify common vulnerabilities.

Output

Review of output

- Common Errors:

  Many instances of the clamav-exec script failing. This could be due to the script not being properly installed or configured, or missing dependencies.

- Specific Findings:

  - 192.168.1.1 (Gateway):
    - http-cookie-flags: Missing httponly flag on SessionID, which is a minor security issue as it might expose the session ID to client-side scripts.

        - Litespeed Web Server Source Code Disclosure (CVE-2010-2333): This indicates a potential serious vulnerability where the server's source code can be disclosed.
        - phpMyAdmin Local File Inclusion (CVE-2005-3299): Potential vulnerability that could allow an attacker to include local files.
    - 192.168.1.4 (ASAttar-ASUS):
        - /server-status/: Directory found which may contain server status information that could be useful for further attacks.

- Recommendations:
    - Fix Vulnerabilities:

        Apply patches and updates for the identified vulnerabilities, especially the Litespeed Web Server Source Code Disclosure and phpMyAdmin Local File Inclusion.

    - Configuration Improvements:
        Set the httponly flag on cookies to enhance security.

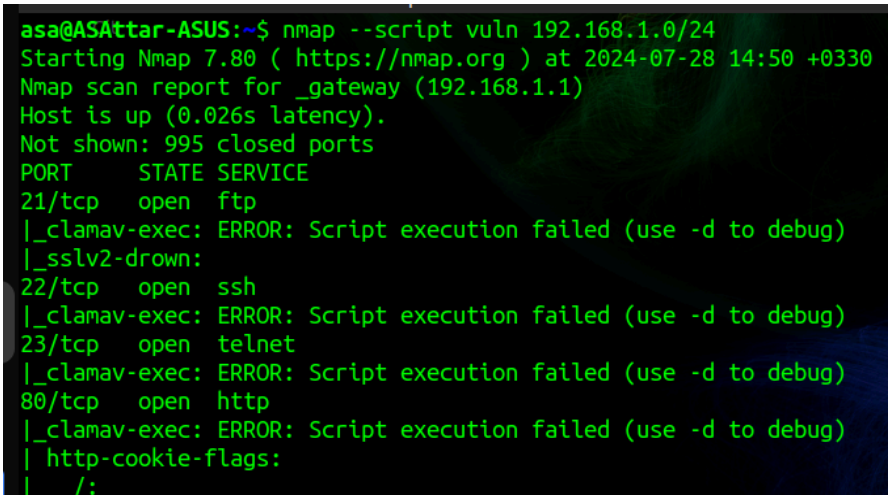
- Script Execution Issues:
    Investigate why the clamav-exec script is failing. This might involve checking for proper installation and configuration of ClamAV and its dependencies.

Port Scanning

determine which ports are open on devices within the network.

```
nmap --script vuln 192.168.1.0/24
```

Explanation: This command scans all 65,535 ports on all devices within the network range 192.168.1.0/24. It helps in identifying services running on non-standard ports.



Output for 192.168.1.1 -> 4 open ports

```
Nmap scan report for ASAttar-ASUS (192.168.1.4)
Host is up (0.000061s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http
|_clamav-exec: ERROR: Script execution failed (use -d to debug)
|_http-csrf: Couldn't find any CSRF vulnerabilities.
|_http-dombased-xss: Couldn't find any DOM based XSS.
| http-enum:
|_  /server-status/: Potentially interesting folder
|_http-stored-xss: Couldn't find any stored XSS vulnerabilities.

Nmap done: 256 IP addresses (2 hosts up) scanned in 36.52 seconds
```

Output for 192.168.1.4 -> 1 open ports

OS and Service Detection

identifying the operating systems and services running on devices within the network.

```
nmap -A 192.168.1.0/24
```

Explanation: The -A flag enables OS detection, version detection, script scanning, and traceroute, providing comprehensive information about devices within the network range 192.168.1.0/24.

```
Nmap scan report for _gateway (192.168.1.1)
Host is up (0.024s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE VERSION
21/tcp   open  ftp
| fingerprint-strings:
|   GenericLines, SMBProgNeg, SSLSessionReq:
|     220 FTP server (192.168.1.1) ready.
|     Unknown command: ""
|   Help:
|     220 FTP server (192.168.1.1) ready.
|     USER expected.
|   NULL:
|_    220 FTP server (192.168.1.1) ready.
22/tcp   open  ssh     Dropbear sshd 0.48 (protocol 2.0)
| ssh-hostkey:
|_  1040 d0:0d:e9:4e:24:4c:41:86:e9:62:32:5c:24:de:3e:4f (RSA)
23/tcp   open  telnet
```

Output for gateway

```
Nmap scan report for ASAttar-ASUS (192.168.1.4)
Host is up (0.000061s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.52 ((Ubuntu))
|_http-server-header: Apache/2.4.52 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works
```

Output for my laptop

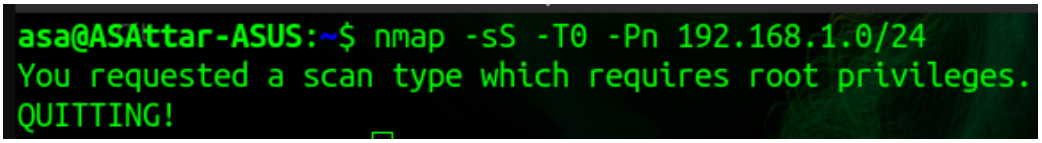
Firewall and IDS Evasion

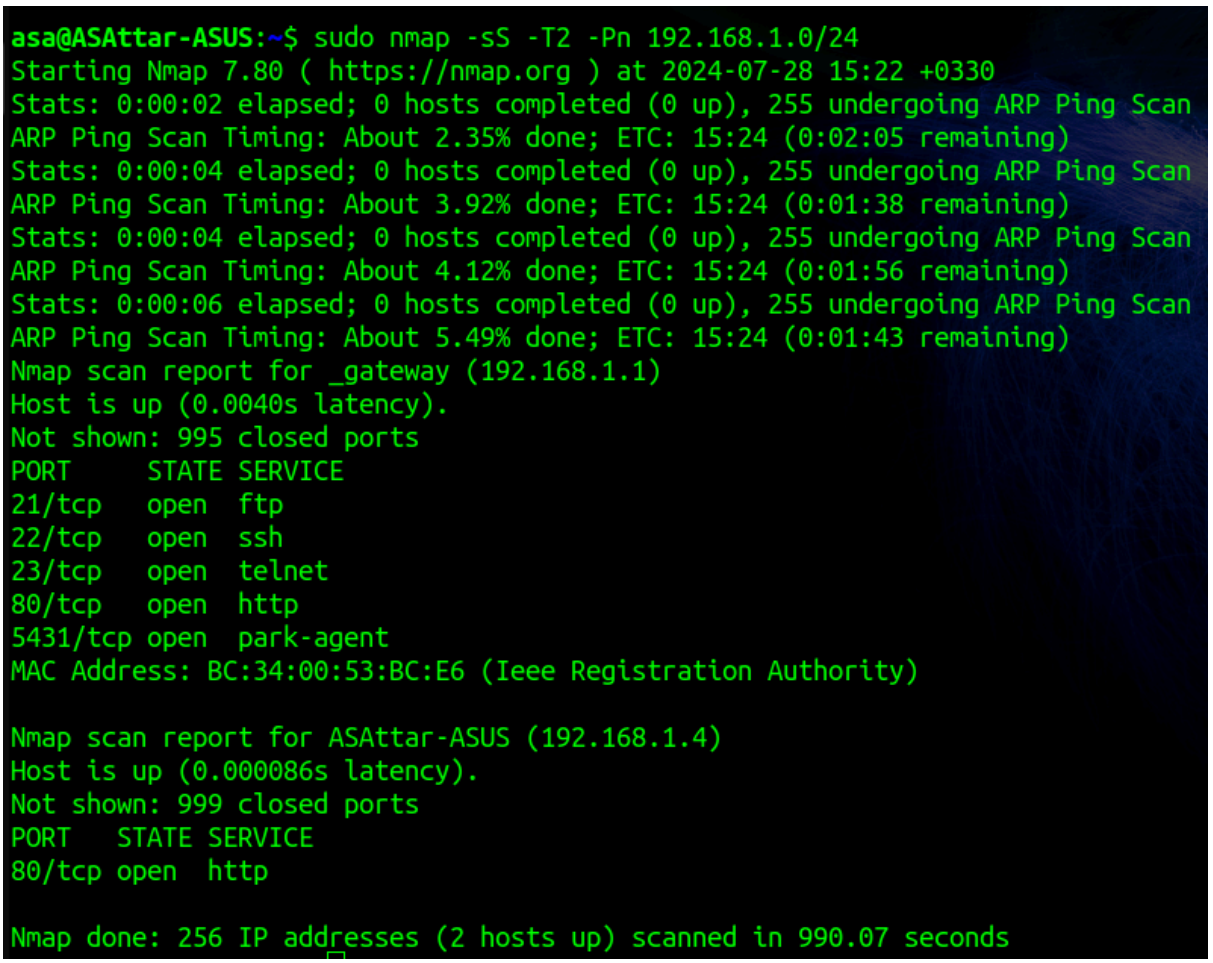
perform a stealth scan to avoid detection by firewalls or Intrusion Detection Systems (IDS).

```
nmap -sS -T0 -Pn 192.168.1.0/24
```

Explanation: The -sS option performs a stealth SYN scan, -T0 sets the timing to "paranoid" to slow down the scan, and -Pn tells Nmap to skip the ping check. This makes the scan less likely to be detected.

Note
this type of scan requires root privilege





Output

Note
I use T2 because of time issues

Scripted Interaction with Hosts

Senario: checking the SSL certificate of web servers on the network for expiry.

```
nmap --script ssl-cert -p 443 192.168.1.0/24
```

Explanation: The --script ssl-cert option runs the ssl-cert NSE script on port 443 of devices within the network range 192.168.1.0/24, retrieving information about their SSL certificates, including expiry dates.

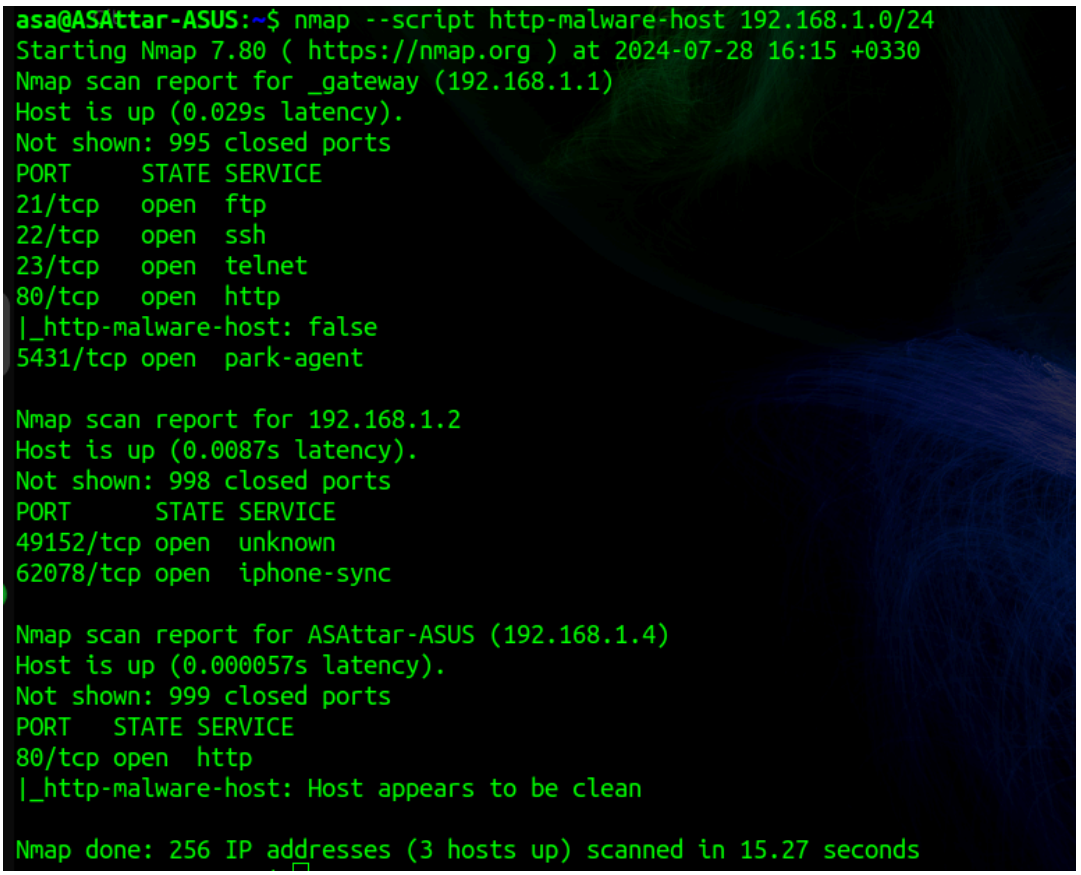
output - the 443 ports are closed for three hosts in network including gateway

Checking for Common Malware Infections

investigating devices for signs of common malware infections.

```
nmap --script http-malware-host 192.168.1.0/24
```

Explanation: The --script http-malware-host option uses a specific Nmap script to check if devices within the network range 192.168.1.0/24 are associated with known malware.



Output

3 hosts are up in the network which the http-malware-host flag is flase or appears to be clean , which makes our network clean.

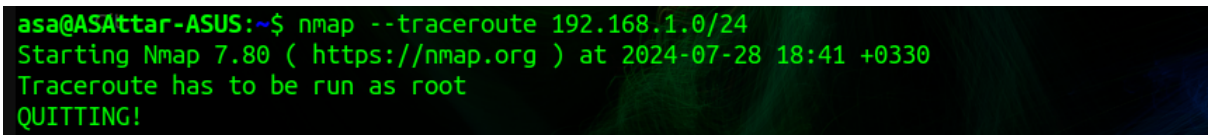
Network Performance and Connectivity Testing

testing the performance and connectivity of a network path.

```
nmap --traceroute 192.168.1.0/24
```

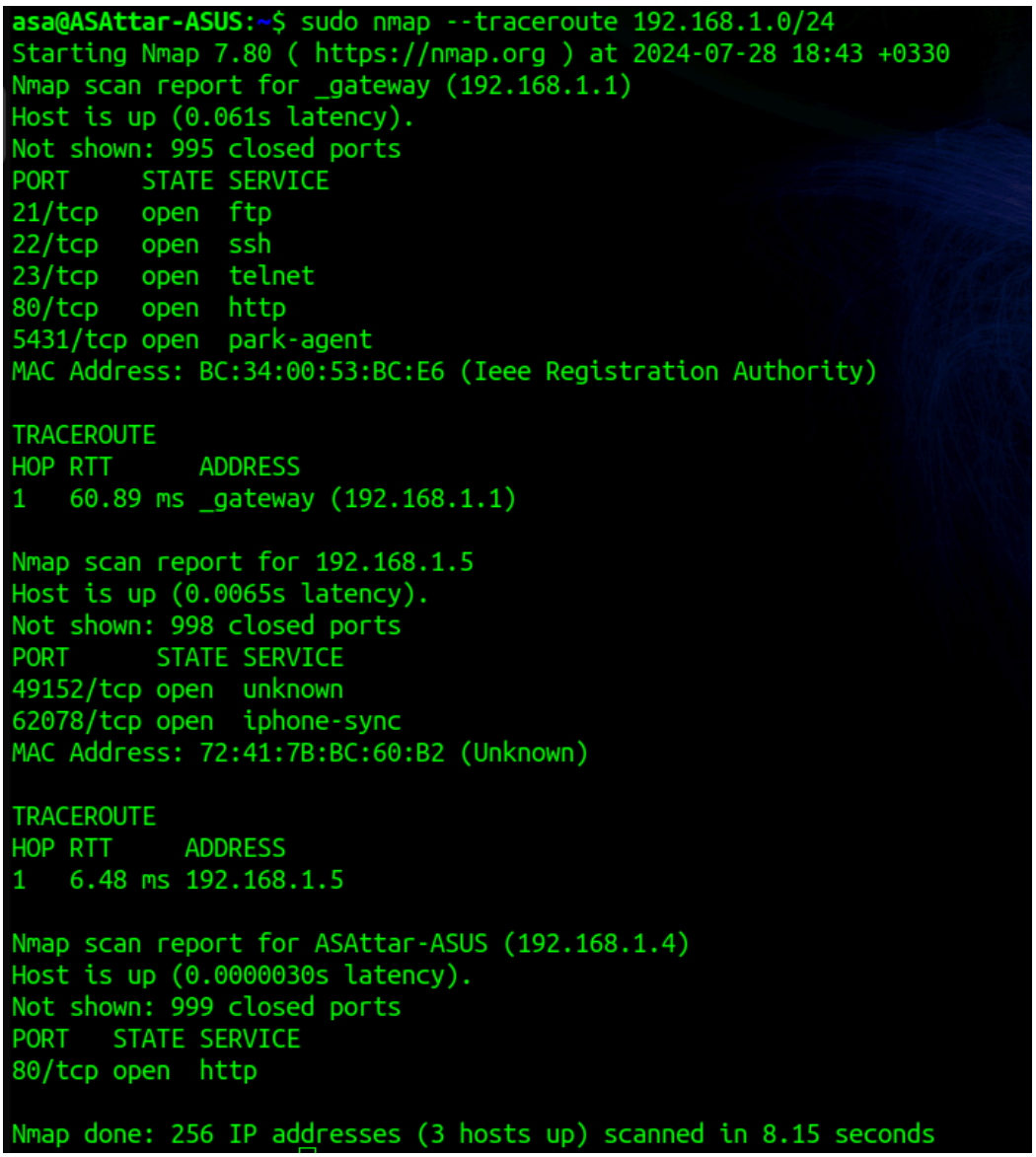
Explanation: The --traceroute option provides information about the path packets take to reach devices within the network range 192.168.1.0/24, which is useful for diagnosing network performance issues.

In summary, Nmap's traceroute function effectively maps the path and counts the intermediary devices (hops) from your device to the target IPs.



Note

The –traceroute option need router privilege to be used



output

- The gateway `192.168.1.1` is confirmed as the router since all devices have it as their first hop.
- The latency (RTT) for reaching the gateway and other devices is relatively low, indicating a local network scan.

Identifying Backdoor Services

checking devices for backdoor services that might have been installed by an attacker.

```
nmap -sV --script=backdoor 192.168.1.0/24
```

Explanation: The -sV option performs version detection, and --script=backdoor runs scripts designed to detect backdoor services on devices within the network range 192.168.1.0/24.

```
asa@ASAttar-ASUS:~$ nmap -sV --script '*backdoor*' 192.168.1.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2024-07-28 19:12 +0330
Stats: 0:00:02 elapsed; 255 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 97.00% done; ETC: 19:12 (0:00:00 remaining)
Stats: 0:00:09 elapsed; 255 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 0.00% done
Nmap scan report for ASAttar-ASUS (192.168.1.4)
Host is up (0.000068s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.52 ((Ubuntu))
|_http-server-header: Apache/2.4.52 (Ubuntu)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 256 IP addresses (1 host up) scanned in 9.07 seconds
```

Use wildcards to match all scripts about backdoor.
Nmap did not find any backdoors on the scanned host. This is a positive indication.

Compliance Auditing

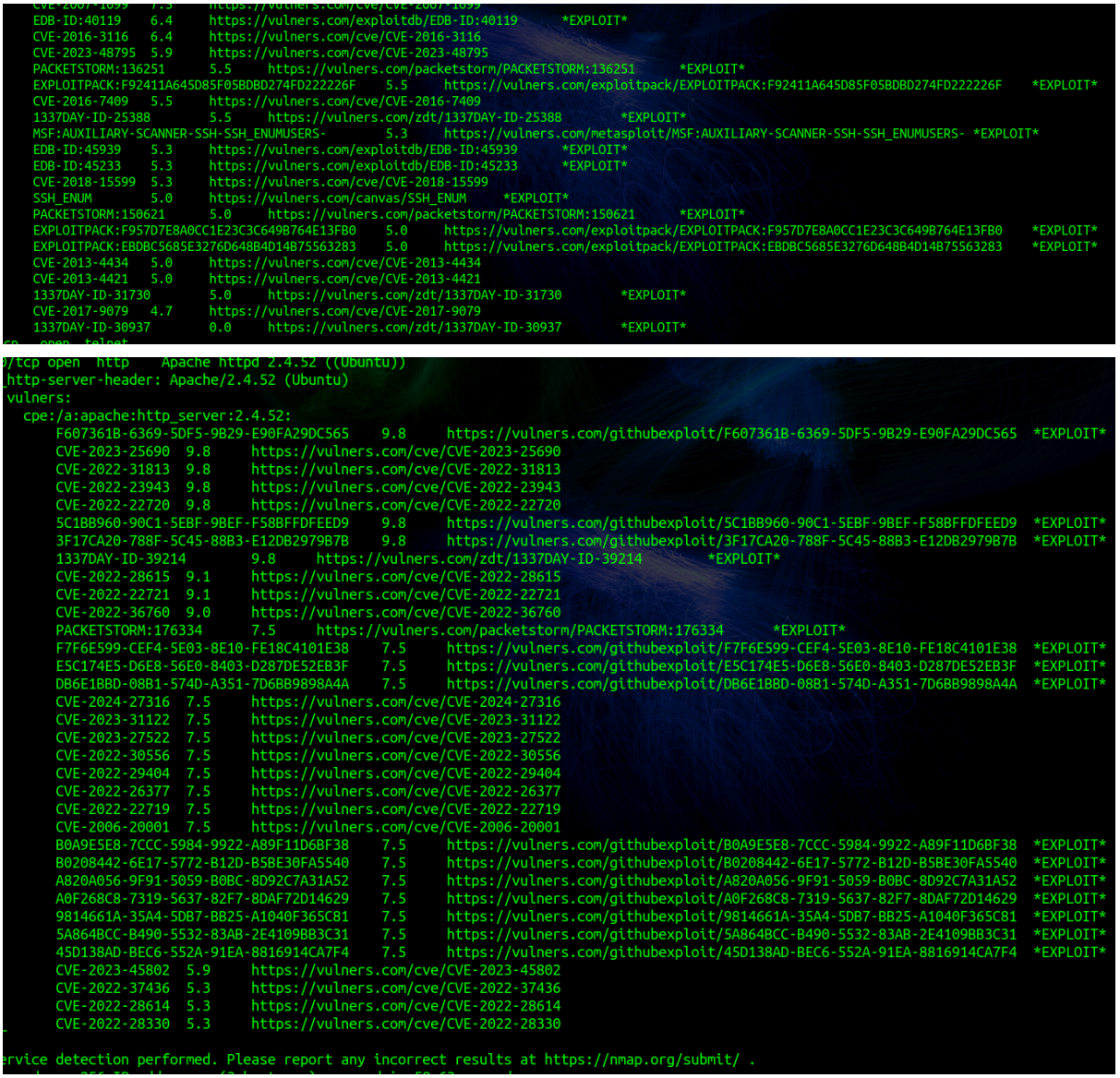
verifing that a network is compliant with specific security standards (e.g., PCI-DSS).

```
nmap --script pcilist 192.168.1.0/24
```

Explanation: The --script pcilist runs an Nmap script to check for compliance with PCI-DSS standards across the network range 192.168.1.0/24.

```
asa@ASAttar-ASUS:~$ sudo nmap -sV --script=*vuln*,*auth*,*safe* 192.168.1.0/24
Starting Nmap 7.80 ( https://nmap.org ) at 2024-07-28 19:25 +0330
Stats: 0:00:02 elapsed; 0 hosts completed (0 up), 255 undergoing ARP Ping Scan
ARP Ping Scan Timing: About 61.57% done; ETC: 19:25 (0:00:01 remaining)
Stats: 0:00:02 elapsed; 0 hosts completed (0 up), 255 undergoing ARP Ping Scan
ARP Ping Scan Timing: About 76.27% done; ETC: 19:25 (0:00:01 remaining)
Stats: 0:00:22 elapsed; 253 hosts completed (2 up), 2 undergoing Service Scan
Service scan Timing: About 60.00% done; ETC: 19:26 (0:00:11 remaining)
Stats: 0:00:32 elapsed; 253 hosts completed (2 up), 2 undergoing Service Scan
Service scan Timing: About 60.00% done; ETC: 19:26 (0:00:17 remaining)
Stats: 0:00:40 elapsed; 253 hosts completed (2 up), 2 undergoing Service Scan
Service scan Timing: About 80.00% done; ETC: 19:26 (0:00:09 remaining)
Nmap scan report for _gateway (192.168.1.1)
Host is up (0.013s latency).
Not shown: 995 closed ports
PORT     STATE SERVICE VERSION
21/tcp   open  ftp
| fingerprint-strings:
|   GenericLines, SMBProgNeg, SSLSessionReq:
|     220 FTP server (192.168.1.1) ready.
|     Unknown command: ""
|   Help:
|     220 FTP server (192.168.1.1) ready.
|     USER expected.
|   NULL:
|_    220 FTP server (192.168.1.1) ready.
22/tcp   open  ssh     Dropbear sshd 0.48 (protocol 2.0)
|_ssh-auth-methods: ERROR: Script execution failed (use -d to debug)
```

Run combination of scripts instead of pcilist because i dont have it in scripts



The output listed the vulnerabilities it detected on each host.

- Bpf security tools

  I use bpftrace which is a high-level language to create and use ebpf traces.
  More specifically implementing bpftrace oneliner.

Tracking Command Execution

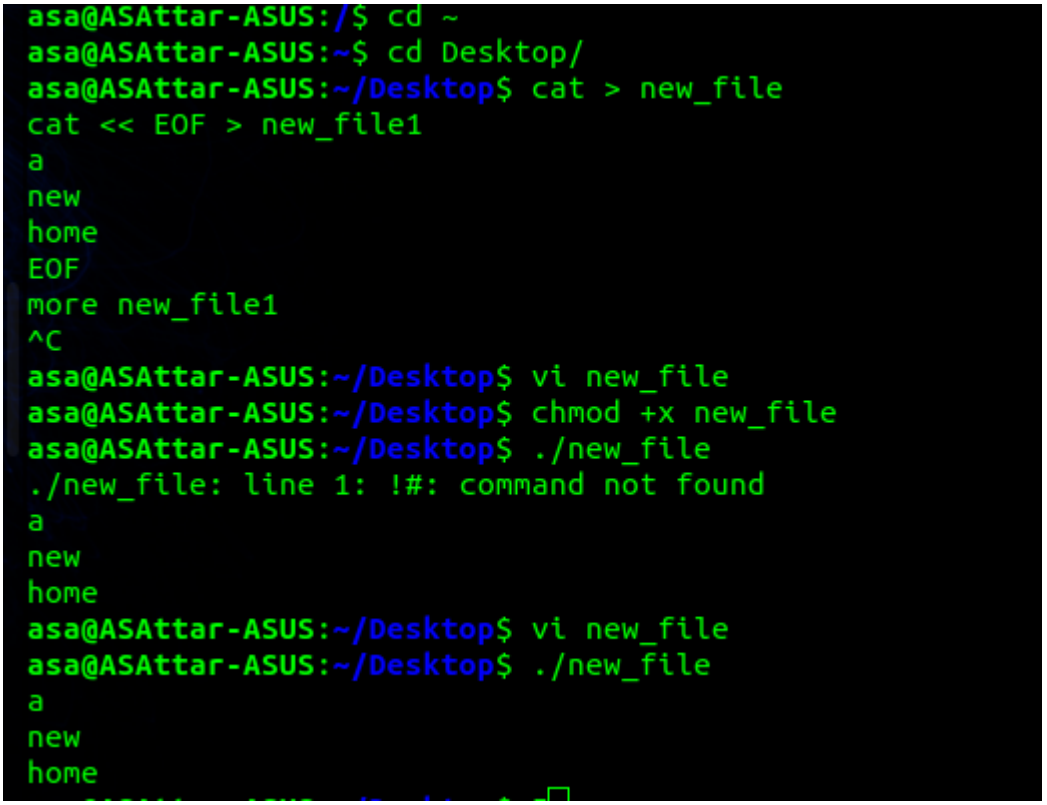
Use Case: Monitor and log every command executed on a system, which can help in forensic analysis in case of a security breach.

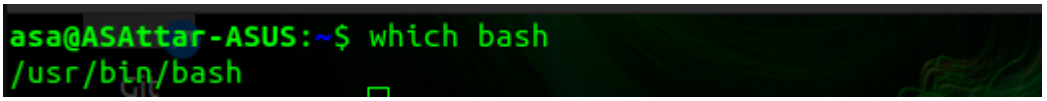
```
bpftrace -e 'tracepoint:syscalls:sys_enter_execve { printf("Command
executed: PID %d, Comm %s, Filename %s\n", pid, comm,
str(args->filename)); }'
```

Explanation:

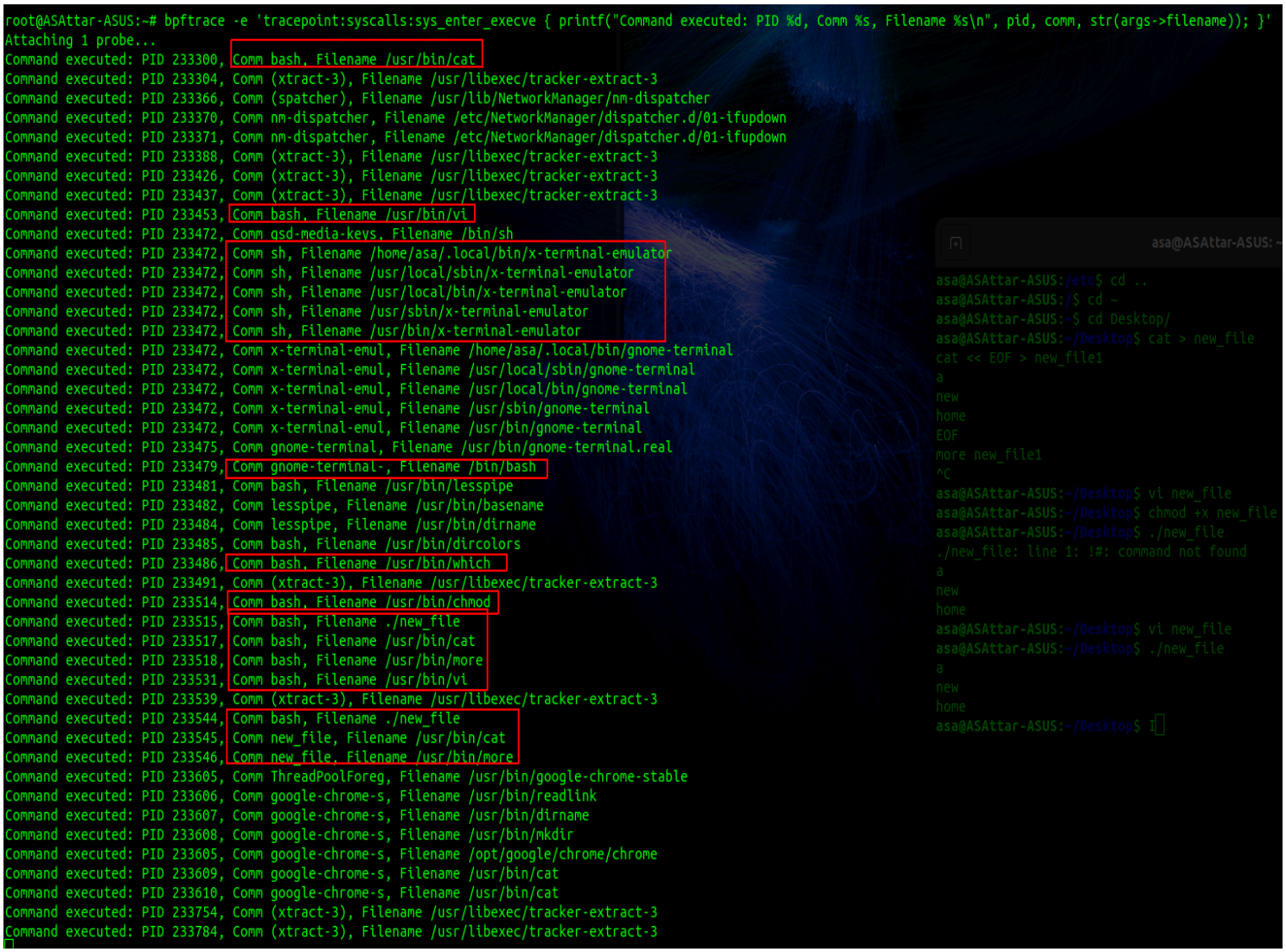
- **tracepoint:syscalls**
  : This tracepoint is triggered whenever a command is executed.
- **printf("Command executed: ...)**: Prints the process ID, command name, and the filename of the executable being run.

Here is the commands i used





Here is the output of trace



Here are some of the known commands i executed

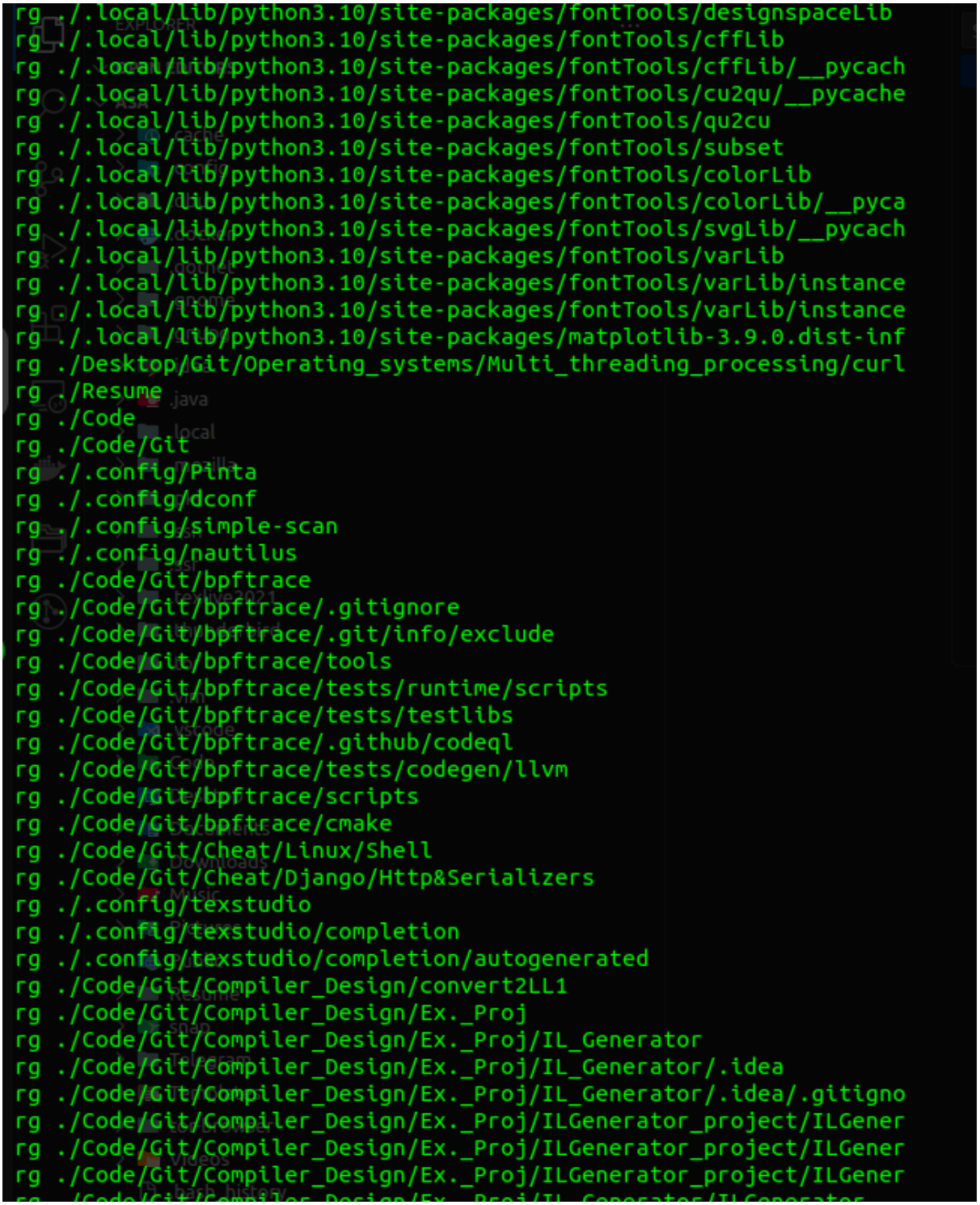
Tracking opening files

Use Case : The script logs each time a file is opened on the system, useful for auditing and monitoring.

```
bpftrace -e 'tracepoint:syscalls:sys_enter_open { printf("%s %s\n", comm, str(args->filename)); }'
```

Explanation :

- **tracepoint**:syscalls: Triggered whenever a file is opened.
- **printf("%s %s\n", comm, str(args->filename))**: Prints the process name and filename being accessed.



Output

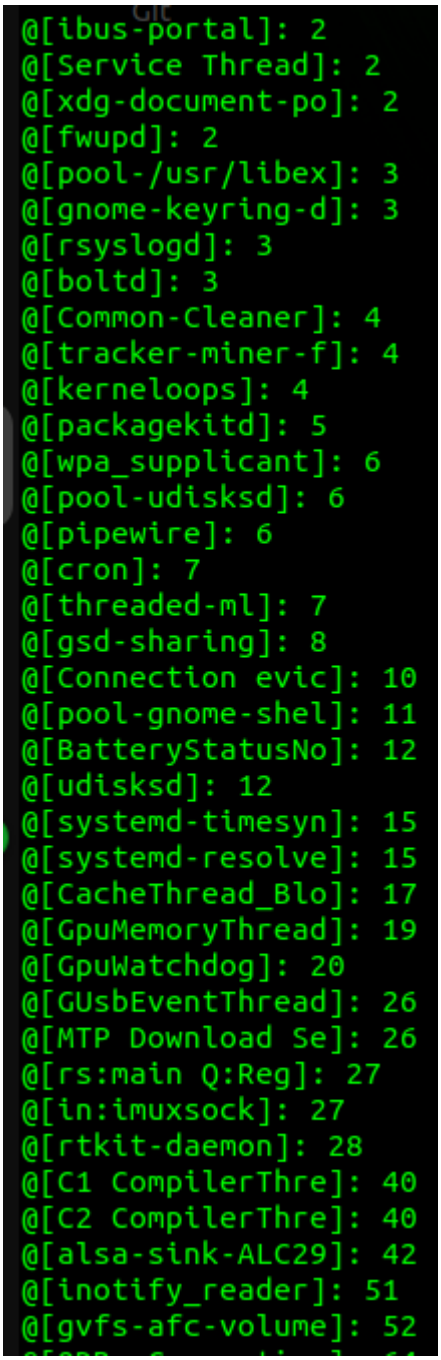
Syscall count by thread name

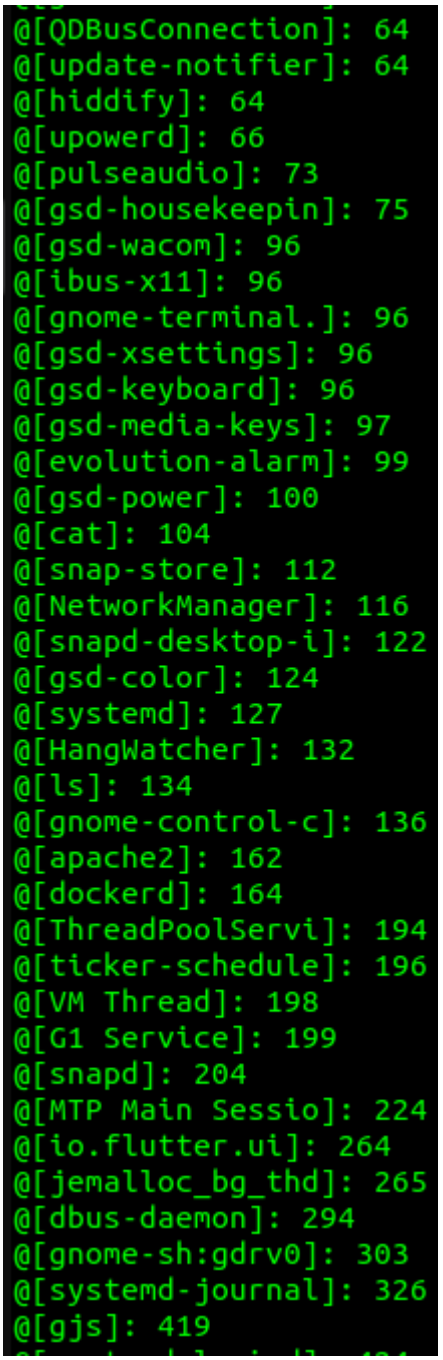
Use case : This `bpftrace` script counts the number of system calls made by each process, which can help in understanding the behavior and resource usage of applications.

```
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @[comm] = count(); }'
```

Explanation:

- **tracepoint:raw_syscalls**
  : Triggered whenever any system call is made.
- **@[comm] = count();**: Aggregates and counts the number of system calls made by each process name (`comm`).





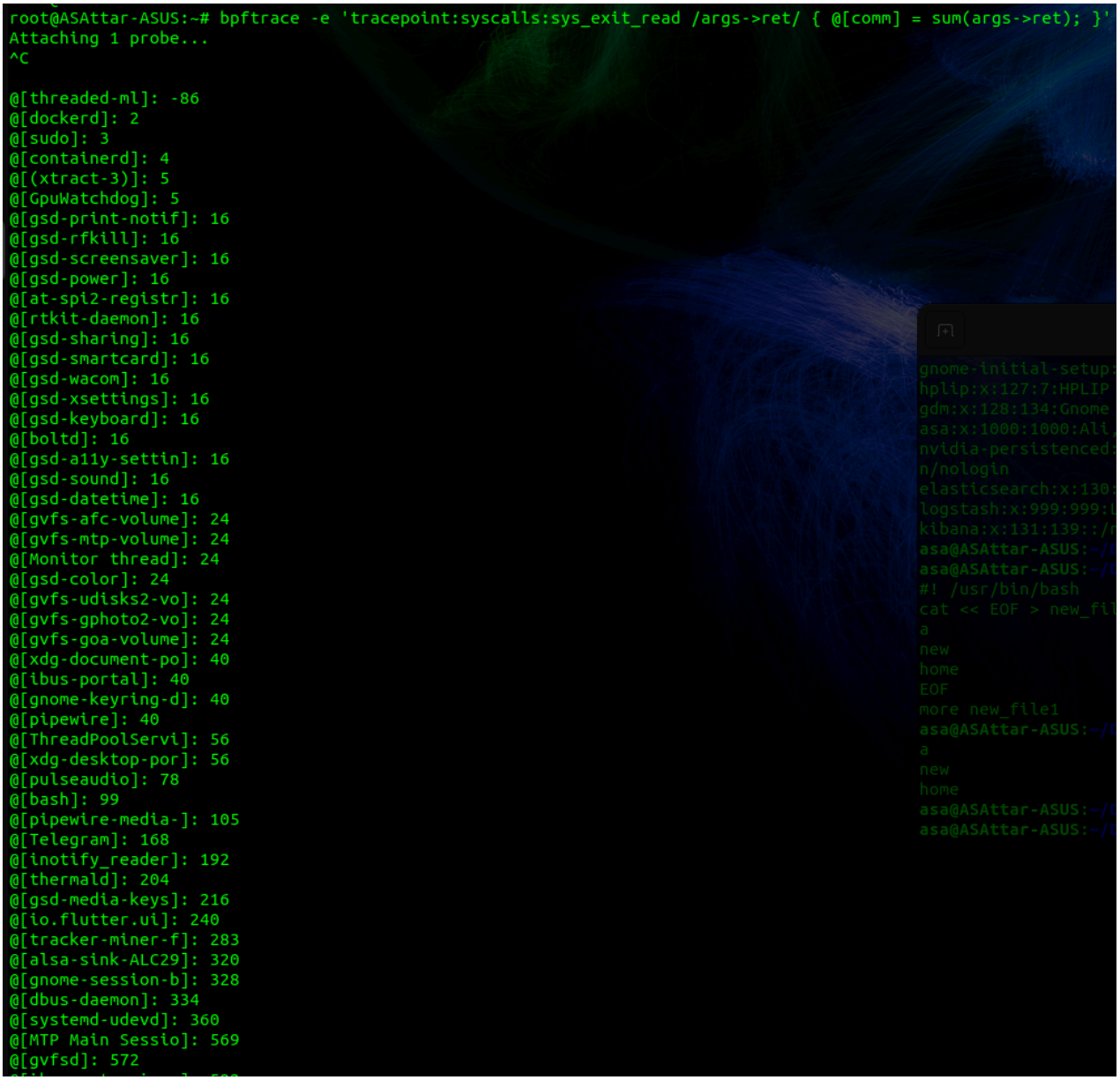
Read bytes by thread name

Use Case : This `bpftrace` script monitors and sums the total number of bytes read by each process, useful for analyzing data usage and identifying processes with high read activity.

```
bpftrace -e 'tracepoint:syscalls:sys_exit_read /args->ret/ { @[comm] = sum(args->ret); }'
```

Explanation:

- **tracepoint:syscalls**
  : Triggered when a `read` system call exits.
- **/args->ret/**: Ensures only successful reads (where `args->ret` is non-zero) are counted.
- **@[comm] = sum(args->ret);**: Aggregates the total number of bytes read by each process, indexed by process name (`comm`).



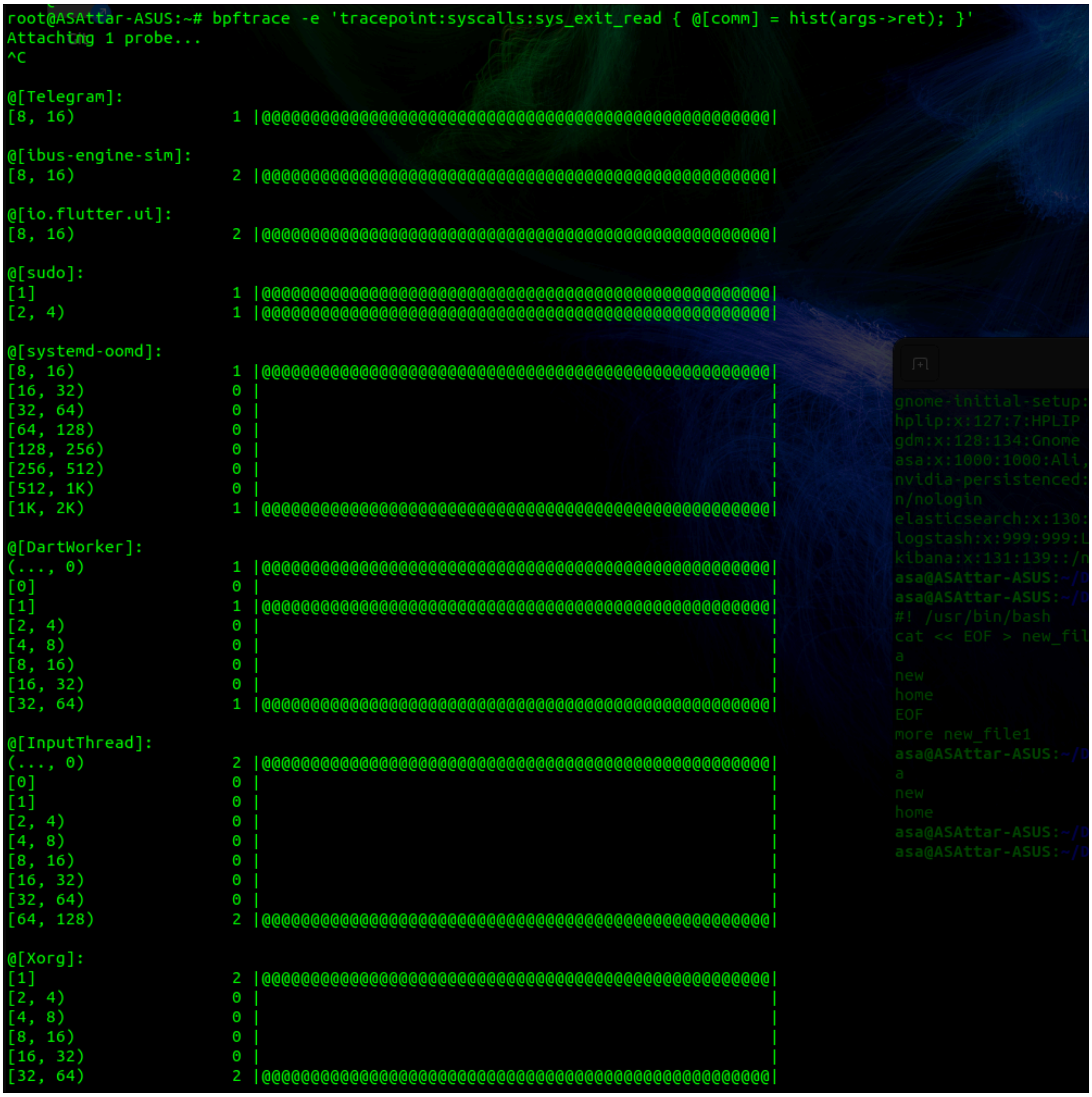
Read size distribution by thread name

Use Case : This `bpftrace` script captures and analyzes the distribution of read sizes for each process, useful for understanding how different processes use read operations.

```
bpftrace -e 'tracepoint:syscalls:sys_exit_read { @[comm] = hist(args->ret); }'
```

Explanation:

- **tracepoint:syscalls**
  : Triggered when a `read` system call exits.
- **@[comm] = hist(args->ret);**: Creates a histogram of read sizes (`args->ret`) for each process (`comm`), showing how frequently different read sizes occur.



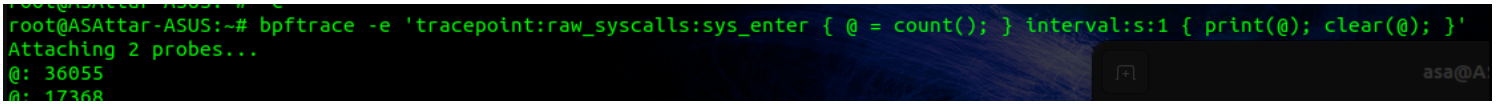
Show per-second syscall rates

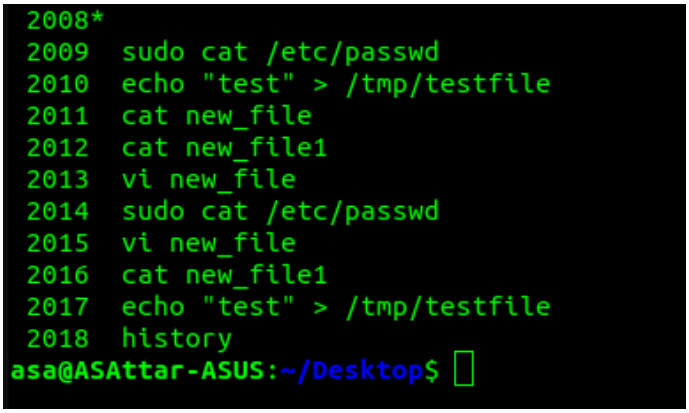
Use Case : This `bpftrace` script measures and displays the rate of system calls per second, which is useful for monitoring system activity and detecting abnormal patterns.

```
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @ = count(); }
interval:s:1 { print(@); clear(@); }'
```

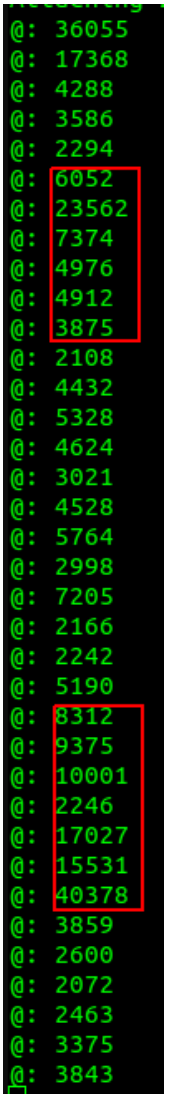
Explanation:

- **tracepoint:raw_syscalls**
  : Triggered every time any system call is made.
- **@ = count();**: Counts the total number of system calls.
- **interval:s:1 { print(@); clear(@); }**: Every second, prints the count of system calls and then clears the counter to start fresh for the next interval.





These point the syscall rates increases because i executed multiple syscalls which each involve more than one syscalls.



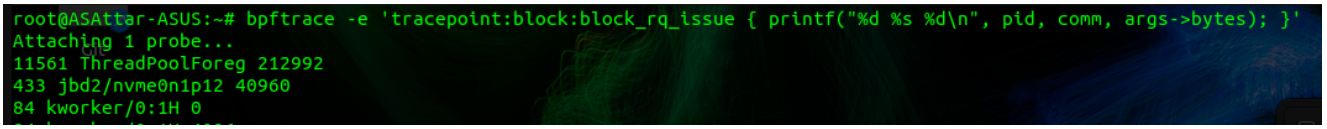
Trace disk size by PID and thread name

Use Case : This `bpftrace` script monitors and logs disk I/O operations, showing the amount of data read or written by each process, useful for analyzing disk usage and performance.

```
bpftrace -e 'tracepoint:block:block_rq_issue { printf("%d %s %d\n", pid, comm, args->bytes); }'
```

Explanation:

- **tracepoint:block**
  : Triggered when a disk I/O request is issued.
- **printf("%d %s %d\n", pid, comm, args->bytes)**: Prints the process ID (`pid`), process name (`comm`), and the number of bytes (`args->bytes`) involved in each disk I/O operation.




```
Attaching 1 probe...
6712 ThreadPoolForeg 73728
660 jbd2/nvme0n1p10 16384
248 kworker/9:1H 0
248 kworker/9:1H 4096
6712 ThreadPoolForeg 4096
254 kworker/0:1H 0
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 12288
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 8192
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
6712 ThreadPoolForeg 4096
254 kworker/0:1H 0
660 jbd2/nvme0n1p10 24576
248 kworker/9:1H 0
```

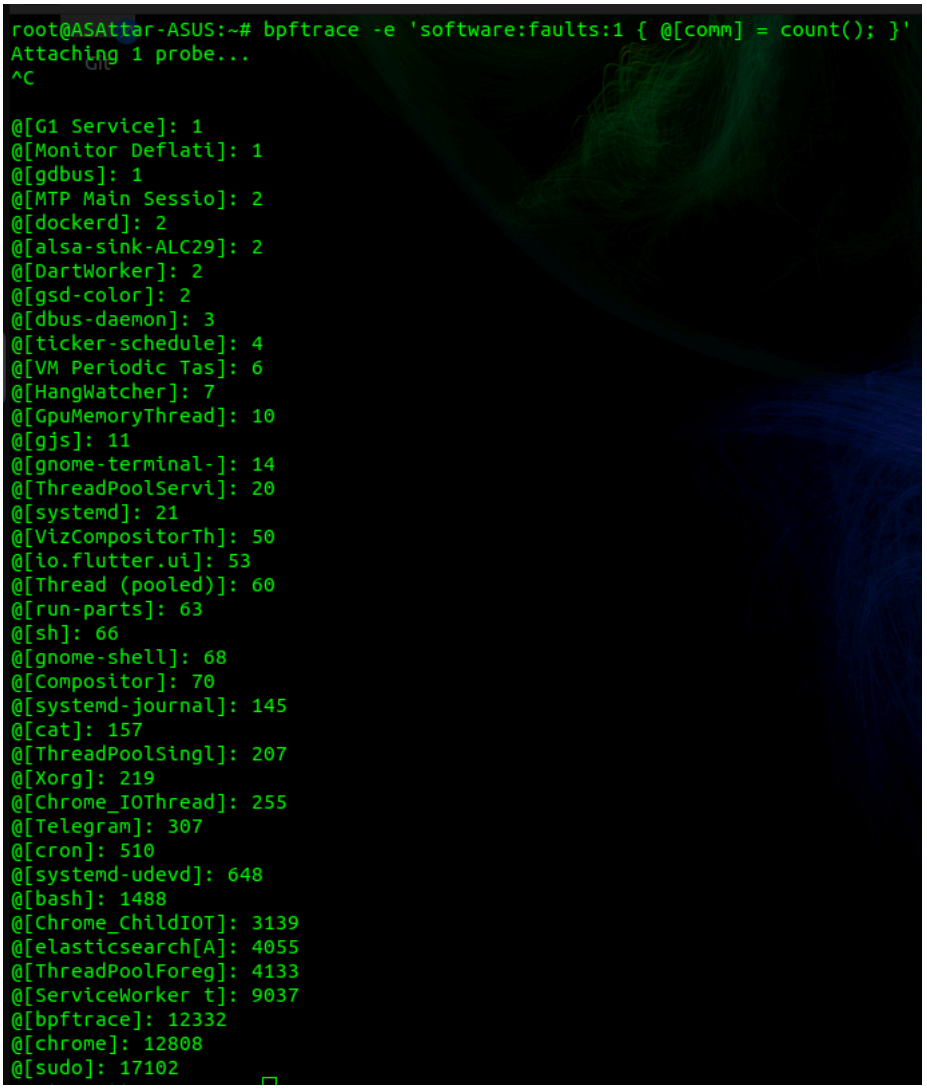
Monitor Page Faults by Process

Use Case : This `bpftrace` script tracks the number of page faults occurring per process, useful for identifying processes with high memory access issues.

```
bpftrace -e 'software:faults:1 { @[comm] = count(); }'
```

Explanation:

- **software:faults:1**: Monitors page faults, which occur when a process accesses memory that is not currently mapped.
- **@[comm] = count();**: Counts and aggregates the number of page faults by process name (`comm`).



We can see that for example chrome has a lot more page faults than the average due to its high consumption of memory

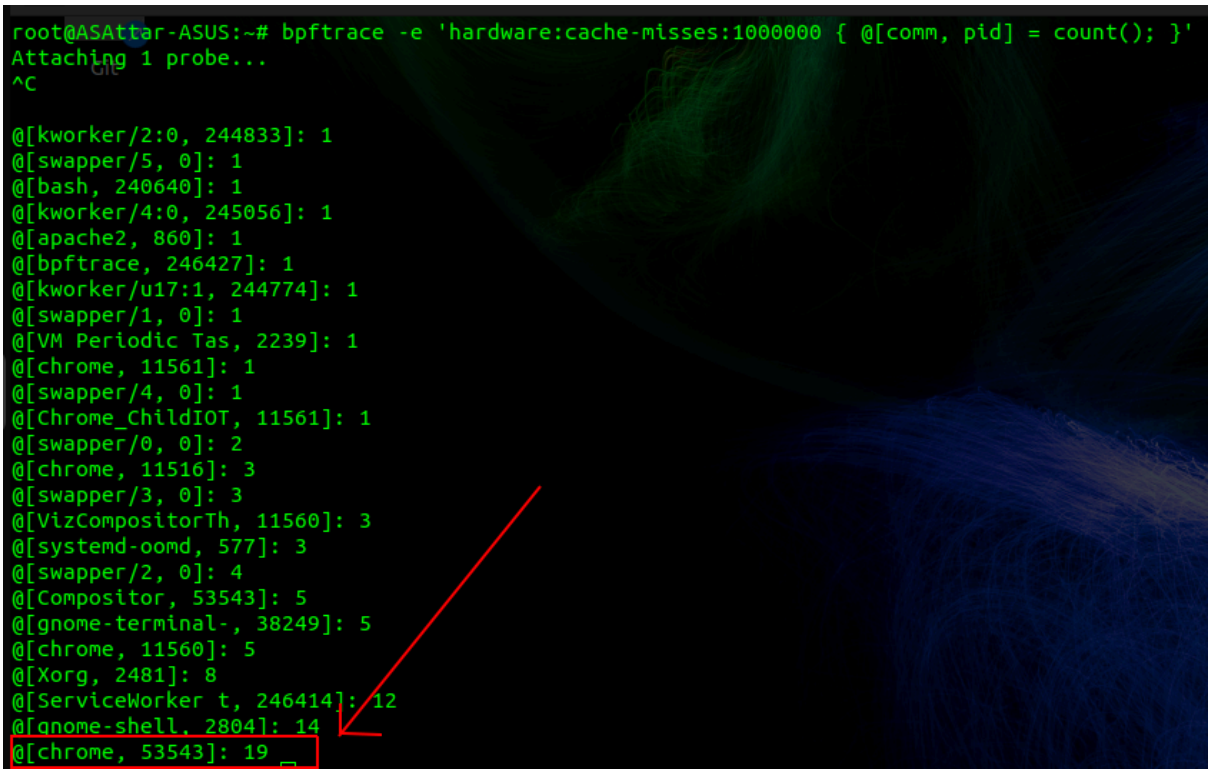
Track LLC Cache Misses by Process

Use Case : This `bpftrace` script counts the number of Last-Level Cache (LLC) cache misses for each process, useful for identifying performance bottlenecks or unusual behavior.

```
bpftrace -e 'hardware:cache-misses:1000000 { @[comm, pid] = count(); }'
```

Explanation:

- **hardware:cache-misses:1000000**: Monitors LLC cache misses, which occur when data is not found in the cache.
- **@[comm, pid] = count();**: Aggregates and counts the number of cache misses by process name (`comm`) and process ID (`pid`).



Here we can see that chrome even has the highest cache misses again because of the same reason as the previous one.

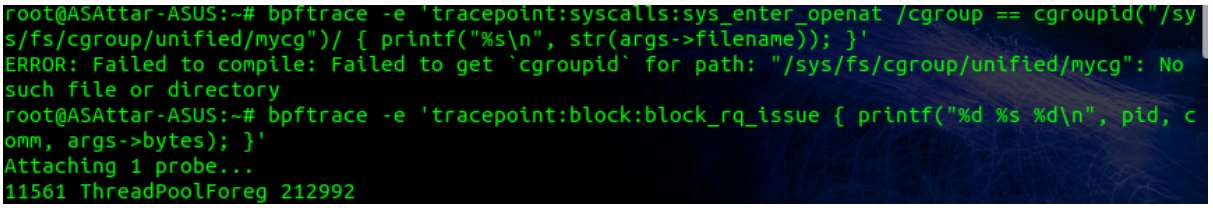
Track Disk I/O by Process

Use Case : This `bpftrace` script logs disk I/O operations, showing the amount of data read or written by each process, useful for analyzing disk usage and performance.

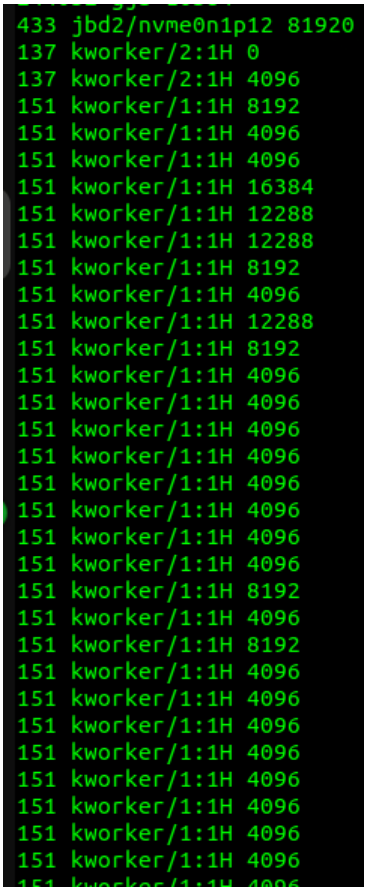
```
bpftrace -e 'tracepoint:block:block_rq_issue { printf("%d %s %d\n", pid,
comm, args->bytes); }'
```

Explanation:

- **tracepoint:block**
  : Monitors disk I/O requests when they are issued.
- **printf("%d %s %d\n", pid, comm, args->bytes)**: Prints the process ID (`pid`), process name (`comm`), and the number of bytes (`args->bytes`) involved in each disk I/O operation.

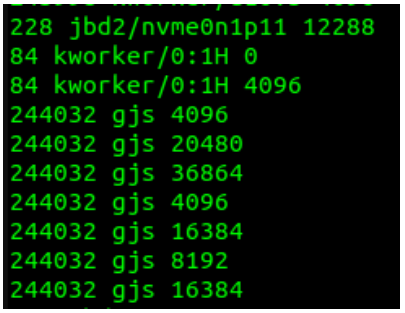


```
root@ASAttar-ASUS:~# bpftrace -e 'tracepoint:syscalls:sys_enter_openat /cgroup == cgroupid("/sy
s/fs/cgroup/unified/mycg")/ { printf("%s\n", str(args->filename)); }'
ERROR: Failed to compile: Failed to get `cgroupid` for path: "/sys/fs/cgroup/unified/mycg": No
such file or directory
root@ASAttar-ASUS:~# bpftrace -e 'tracepoint:block:block_rq_issue { printf("%d %s %d\n", pid, c
omm, args->bytes); }'
Attaching 1 probe...
11561 ThreadPoolForeg 212992
```

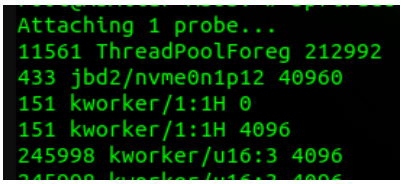


```
433 jbd2/nvme0n1p12 81920
137 kworker/2:1H 0
137 kworker/2:1H 4096
151 kworker/1:1H 8192
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 16384
151 kworker/1:1H 12288
151 kworker/1:1H 12288
151 kworker/1:1H 8192
151 kworker/1:1H 4096
151 kworker/1:1H 12288
151 kworker/1:1H 8192
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 8192
151 kworker/1:1H 4096
151 kworker/1:1H 8192
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
151 kworker/1:1H 4096
```

```
asa@ASAttar-ASUS:~/Desktop$ cat > newjfksl
fjklsd
jfsldk
jfsl
jfsalk
jfsdla
jfksld
^C
asa@ASAttar-ASUS:~/Desktop$ more newjfksl
fjklsd
jfsldk
jfsl
jfsalk
jfsdla
jfksld
asa@ASAttar-ASUS:~/Desktop$ more newjfksl
fjklsd
jfsldk
jfsl
jfsalk
jfsdla
jfksld
asa@ASAttar-ASUS:~/Desktop$ more newjfksl
fjklsd
jfsldk
jfsl
jfsalk
jfsdla
jfksld
asa@ASAttar-ASUS:~/Desktop$ more newjfksl
fjklsd
jfsldk
jfsl
jfsalk
jfsdla
jfksld
asa@ASAttar-ASUS:~/Desktop$ more newjfksl
fjklsd
jfsldk
jfsl
jfsalk
jfsdla
jfksld
asa@ASAttar-ASUS:~/Desktop$ more newjfksl
fjklsd
jfsldk
jfsl
jfsalk
jfsdla
jfksld
```



```
228 jbd2/nvme0n1p11 12288
84 kworker/0:1H 0
84 kworker/0:1H 4096
244032 gjs 4096
244032 gjs 20480
244032 gjs 36864
244032 gjs 4096
244032 gjs 16384
244032 gjs 8192
244032 gjs 16384
```



```
Attaching 1 probe...
11561 ThreadPoolForeg 212992
433 jbd2/nvme0n1p12 40960
151 kworker/1:1H 0
151 kworker/1:1H 4096
245998 kworker/u16:3 4096
```

Some of the rest of the logs

The left result (repeated logs of process 4096) is all because of the write and read operations i was doing simultaneously.