CSCI 5573: Advanced Operating Systems
Fall 2024

Programming Assignment Three

Due Date: 12/06/2024

1. Inserting a new system call
   Extend *capsule* that you built in Programming Assignment Two to allow processes in a capsule to connect to the Internet as well as have processes in different capsules communicate with one another. To test this functionality, write a program that connects a process in a capsule to some server in the Internet. Write another program that establishes a TCP client-server communication between two processes in two different capsules running on (a) the same host, and (b) different hosts.

2. Simple programming using eBPF

   a) Write an eBPF program to count incoming packets per source IP address. Your program should print out source IP address along with count every time a packet arrives.

      Steps:
      - Use XDP to detect incoming packets
      - Parse ethernet/IP headers to get to the source IP address
      - BPF hash map to store and atomic increments.

   b) Implement an eBPF-based program to detect and log SYN flood attempts. Send a custom alert to the userspace using the events table.

      Steps:
      - Similar to the above, but now we need to distinguish between TCP SYN/ACK packets
      - Using BPF_PERF_OUTPUT and the events table

3. Build a load balancer in kernel using eBPF

   (a) Create an eBPF program that functions as a basic layer 4 load balancer. It should redirect incoming packets to a pool of backend servers based on a round-robin algorithm.

      Steps:
      - Store IPs in a BPF array
      - Use a round robin algorithm for load balancing. Make sure to recalculate IP checksum
      - Retransmit the packet.

   (b) Modify the load balancer from the previous task to use a consistent hashing algorithm based on source IP to ensure session persistence. Incoming packets from the same client IP should always be directed to the same backend server.

      Steps:
      - Maybe try using bpf_ntohl, then use a simple % operation for consistent hashing.

CSCI 5573: Advanced Operating Systems
Fall 2024

4. Build a container monitoring tool using eBPF

   a) Write an eBPF program that attaches to container start events by tracing the clone system call with specific flags indicating the creation of new namespaces (e.g., CLONE_NEWNS, CLONE_NEWPID). The program should capture the PID, command name, and namespace identifiers of the new container processes and expose this information to user space.

      Steps:
      - Kprobe(SEC("kprobe/clone"))
      - eBPF events

   b) Develop an eBPF program that monitors all system calls made by a specific container identified by its cgroup or PID namespace. The program should log the system call names and PIDs of processes within the container.

      Steps:
      - SEC("tracepoint/syscalls/sys_enter")
      - bpf_get_current_task();

   c) Create an eBPF program that tracks the network bandwidth usage (both ingress and egress) for each running container on the system. The program should associate network packets with containers using cgroups or namespaces and expose bandwidth metrics to user space.

      Steps:
      - SEC("tracepoint/net/net_dev_queue")
      - SEC("tracepoint/net/netif_receive_skb")
      - task->nsproxy->net_ns->ns.inum to get ns