Google

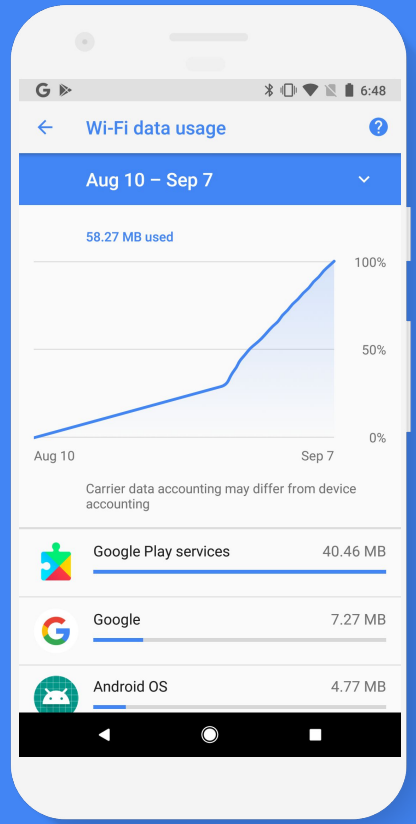
eBPF cgroup filters for data usage accounting on Android

Linux Plumbers eBPF microconference
September, 2017

Chenbo Feng <fengc@google.com>
Lorenzo Colitti <lorenzo@google.com>

What is xt_qtaguid?

- Network traffic monitoring tool on Android devices
- Replaced the xt_owner module inside android device kernels
- Counting packet against the correct app uid.
- Filtering per-app traffic with socket owner match



Xt_qtaguid module

Problems with current module

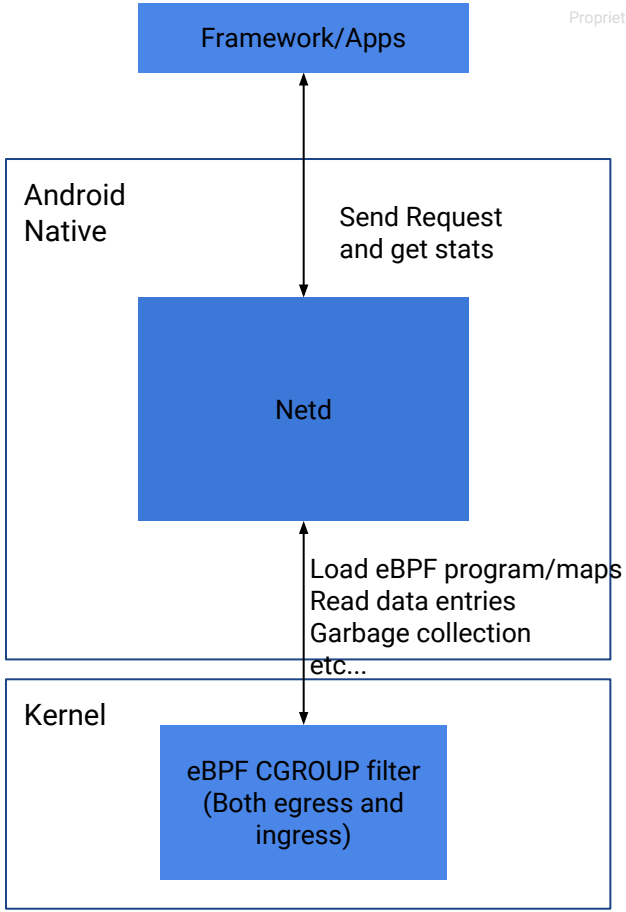
- Totally out of linux kernel tree and not upstreamable.
- The version of this module varies with kernel version.
- Stability, maintenance, and soon performance issues.

Goal

- developing a new tool to realize similar function as xt_qtaguid module with no out-of-tree code

Basic Design

- Per-cgroup eBPF program to perform accounting
  - Ingress: Transport layer (e.g. tcp_v4_rcv), same as eBPF socket filter
  - Egress: Network layer (eg. ip_finish_output)
- Stats received are stored in eBPF maps.
- Stats periodically retrieved by privileged process from eBPF map
- Apps tag sockets by sending fd using binder call to privileged process



Progress so far

- Fixes for accounting correct packets
- New getsockopt SO_COOKIE
- Helper functions to get UID and cookie
- All upstream as of 4.12, backported to android-4.9
- In progress: LSM hooks and selinux checks for eBPF operations

eBPF Challenges

- Memory management
  - xt_qtaguid can call kmalloc
  - eBPF maps cannot be resized, consume unswappable kernel memory
    - Tagging socket can fail, but not being able to account traffic to UID unacceptable
  - Need garbage collection in userspace to run concurrently with kernel program
- Security model not fine-grained
  - Everyone can write to maps and load programs (bad)
  - Only CAP_NET_ADMIN can write to maps, so processes can’t tag own sockets

Implementation Challenges

- Cgroup eBPF program call sites scattered around kernel
  - xt_qtaguid simply uses the netfilter hooks, which already have to cover all codepaths
  - eBPF, needed several fixes to ensure different types of packets were counted [only] once
    - Still can’t count IPv6 SYN+ACKs
    - Not sure how to count IPsec packets yet
- Split user/kernel space solution
  - Many moving parts: kernel program, netd, init, …
  - Concurrent access to cross-map values between user and kernel space
    - No locks, and can’t lock between kernel and userspace since netd can sleep
  - Need to deal with netd crash recovery

THANK YOU

Q & A

Android socket tagging

- Semantics:
  - Counts packets and bytes on combination of app, app-defined tag, interface
  - Allows assigning 64-bit tag to every socket
    - Socket tags comprised of 32 bits UID (i.e., app) and 32 bits app-defined tag
    - Privileged UIDs may impersonate other UIDs (e.g., download manager billing traffic to app that requested the download)
- Userspace interface:
  - Apps tag their own sockets using /proc interface
  - System collects data by scraping /proc

Why cgroup filtering?

Following alternatives considered cannot fulfill our needs

xt_ebpf with pinned eBPF object

- skb->sk usually unavailable on ingress side

Per-socket eBPF filter

- Only does input packets
- Need to apply program to every fd individually
- Some sockets don't have an fd, so can't attach program to them

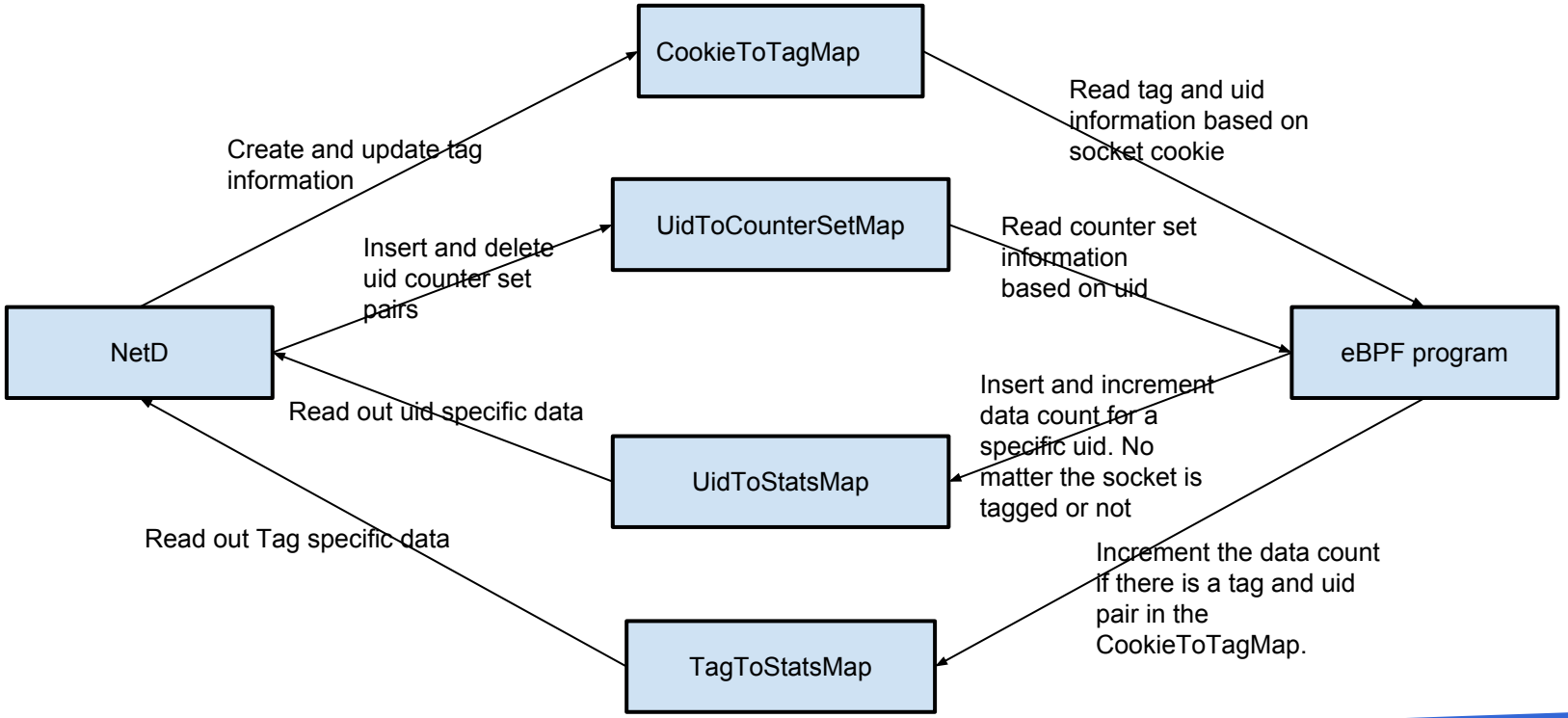
tc bpf

- Only does output packets

Data structures

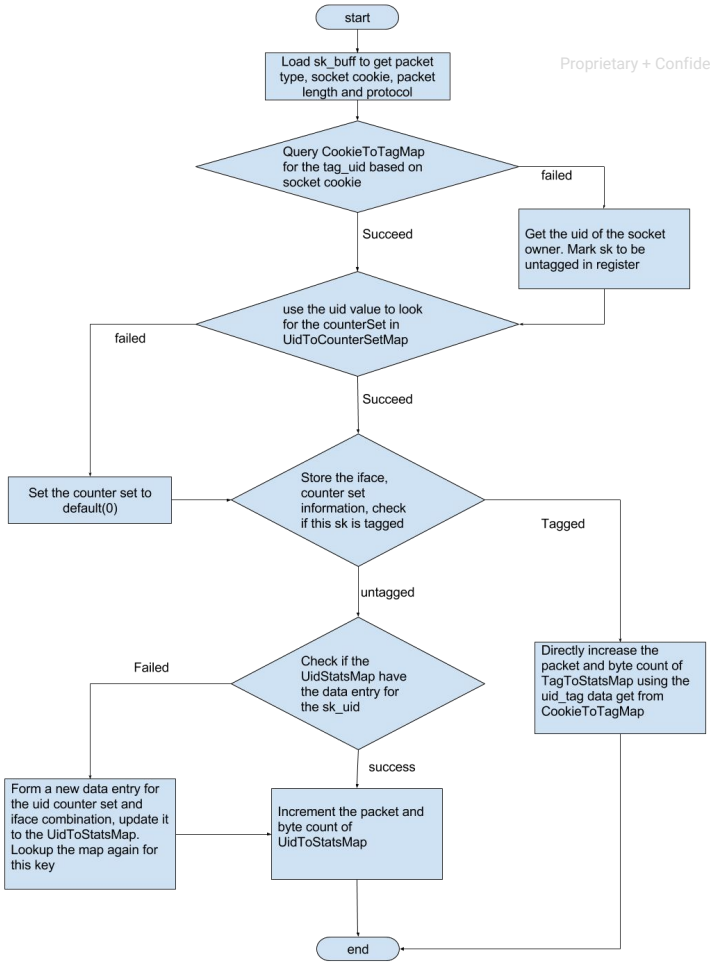
- Use sk_cookie to identify socket in various EBPF maps
  - If empty, cookie initialized by eBPF program when a packet is processed
- Cookies mapped to:
  - Socket IDs (uid | tag) if socket is tagged
- Stats entries are mapped with two struct
  - Key struct contains Socket ID | foreground state | interface
  - Value struct contains tx/rx packets number and tx/rx bytes
- Overall stats are in UidToStatsMap
- Tagged sockets stats are in TagToStatsMap

Userspace kernel interaction



Kernel Program

- Written in assembly like instruction arrays
  - libelf is GPL and is not compatible with Apache
  - Potentially allow creating eBPF program at run time.
- Loaded into the kernel on netd startup
- Packet information collected:
  - Socket uid
  - Packet type (tcp, udp, other)
  - Packet length
  - rx/tx interface



Security Model

- Adding LSM hooks and selinux checks for eBPF operations in progress
- Selinux is responsible for restricting the access to eBPF object and cgroup.
  - Only allow netd to create eBPF maps, update element and load eBPF program
  - Only allow netd to access file under bpf filesystem
  - Only allow netd to access the root directory of cgroup v2
- May allow system server directly read maps to enhance performance