LSF/MM/BPF Raw Meeting Notes

(raw / wip note collection from Joe Stringer)

Monday May 2, 2022

Cgroups BPF metrics

Hao Lu - Google
09:30 Monday

- Gathering cgroupfs stats via bpf. Couple of options:
  - BPF map, read from map
  - BPF iterator
- Difficulty: Unprivileged users create cgroups, so no way to initialize the BPF observability side since BPF requires privileges.
- Solution space:
  - (1) User requests for BPF plumbing to occur via local daemon.
    - Problems: Additional load on node agent; No HA if cgroups are constantly created
    - John: Nested cgroups, no way for inner tenant to request this from the outer. Inner is handled by application team, outer is handled by security team.
  - (2) BPF program monitors cgroup create, then attach BPF programs
    - Helpers for calling into the BPF syscall logic from another BPF program
  - (3) Fine-grained ACLs for BPF syscalls
    - Per-program, per-operation permissions delegated to unpriv users
    - (group stats iter, link create), (stats iter link, obj_pin)
    - Why not just support this case with existing unprivileged model?
      - Alexei mention: Loading BPF programs must stay root. Also link_create (new fds), but other BPF access could become unprivileged, managed by FS permissions on BPFFS
- Related: Daemon-less container engine
  - Idea: Delegate container runtime stuff to sleepable BFP tracepoint programs?
  - Feedback from room: Why? (Though also why not? Possible but interested in motivation)
    - Reduce the surface of privileged processes.
    - KP: Tradeoff here on security perspective. Converting to daemonless says that the surface may increase due to unprivileged processes now triggering this logic. Cannot carry over understanding of security model from existing monolith programs.
- Side-note on local storages

    - Cgroup local storage <-> BPF program there is a 1:1 limitation
    - Socket local storage <-> BPF program supports more than one storage at time
    - Why? Cgroup local storage was written first, probably just historic

BPF Dynamic Pointers

Joanne Koong - Meta
11:00 Monday

- New helpers for creating a dynamic pointer buffer by calling with pointer + length
- Verifier can use the length to verify subsequent accesses into this pointer are valid
- This way, at runtime you don't need to do bounds checks constantly to inform the verifier that the accesses are OK.

BPF for debugging

Song Liu, Meta
11:30 Monday

Goal: Enable debugging with BPF, but with more constrained security than CAP_BPF
Options

- Pre-determine BPF programs for unpriv users to use (inflexible)
- Allow tracing of own events in own context
    - Brendan: async tasks aren't tied to the current context / task /etc.
        - That said, can always walk up the contexts to get access to the events you need access to
- Something about a model where a root privileged process manages a map that provides socket references, then there's a filter on access from unpriv into this map.
- Filtering both which tracepoints and also which arguments are able to access
    - fentry trampoline
    - sched_switch - can contain info about other context in the args
    - Verifier generate a matrix of flags for access
    - When attaching a program, use this matrix to apply a policy

Cross-platform libbpf & bpftool

Dave Thaler, Microsoft
13:00 Monday

- Considering libbpf as the API/ABI boundary. Big difference on Windows: HANDLE (pointer size) is the kernel equivalent to fd (uint). Windows commonly uses fd in userland, but the kernel only knows handles. libc equivalent allows user programs to use fd, then converts these to HANDLE before ioctl()

Sockmap updates and new features

John Fastabend, Isovalent
14:00 Monday

- Generic L2 bpf attach hook
    - Hook into cgroups or into netns?
    - Netdevice creation time, hook in there
    - Sleepable programs hook at link create? (Kumar)
        - BPF_LINK attach directly to device
        - As long as the host daemon keeps the fd, then the attachment stays tied to the device, so no way for the pod to unload the program
        - TC-like hook, but not actually TC. No qdisc, no ingress/egress/etc.
        - Alexei: Kernel side seemed pretty close. Libbpf was harder to agree on.
    - Some discussion on if you have the device, you can map that to cgroup/ns
- Sockmap
    - Add psock_ctx to this sk
    - Previous design: Switch-inspired big hash table to make LB decisions
    - Goal: Eliminate the map, (...) support use cases that don't need to LB.
    - Unbalanced API: tx is all sockets, rx is all skbs
    - Move rcv to use scatter/gather list directly
    - Tail call recursion limit
        - Iterative loop to avoid this
    - For now, just use from established socket hooks. Later, move to socket creation time (would need new hooks)
        - Daniel: Does this work for unconnected sockets
        - John: CI, but unclear about production usage yet.
    - Listen BPF -> Establish doesn't inherit programs to the accepted socket
        - No fast open hook (not sendmsg)

BPF for "standard interface" flexibility

KP Singh, Google
14:30 Monday

- Problem, configuring file interfaces (eg in cgroupfs or procfs) backed by eBPF
    - "People" don't like that there would be non-standard random files in here
- (Hao) Solutions
    - Pin a BPF object into the cgroupfs
        - Then you can open, read, write, etc.
    - Replicate cgroupfs in bpffs
        - Link back to concerns from first talk of today

        - Task_iter for attaching programs, considered earlier from Yonghong. Some concerns were raised, didn't go through.
        - Overlayfs combining cgroupfs and bpffs, earlier proposal.
            - cgroupfs underneath, bpffs on top
            - Underlying layer (cgroupfs) changes from time to time. Overlayfs requires underlying filesystem to be immutable. Undefined behaviour.
                - Create cgroup, then delete, file is cached in overlayfs.
                - Inconsistent view.
                - Sounds like a question for FS track. Is it possible to improve overlayfs to support this case.
                - More flexible. Even if it's limited to pseudofs, avoid the caching layer for these.
- Can we add this under some expert flag? Who can sign off on this?
    - That's not the Linux philosophy.
- Some mention about inheritance problems - child cgroups should inherit the BPF progs from the cgroups with the upper BPFFS

DIGLIM

Roberto Sassu - Huawei Dresden Research Center
15:30 Monday

- Discussion about the "Conversion stage" - referencing some controversial argument over the design during the BPF conf call.
    - Don't have to agree on the model to be able to agree that there should be a helper to access the keyring
    - Some discussion on how to make it future-proof - which keyring for example.

Tuesday May 3, 2022

Program Signing

Jason Baker - Google
09:00 Tuesday

- Identity-based policy, granular policies
    - Identity: Root key, similar to SSH key format? Principles, "this is allowed to run as..".
- What do we need in BPF to support this?
    - One notion: The kernel enforcement point should be a BPF program
        - Identity may vary from use case to use case. May not be one canonical view on what "Identity" means.

        - We can then evolve over time separately or eventually likely build some common component to live there.
- Signature verification through a early-boot + frozen hook with some sort of helpers.
- Multi-platform: some sort of gatekeeper / dynamic mechanism is preferred, avoid hard-baking this into the kernel. Some notion that the signatures could vary for example.
- Strong preference to sign the Capabilities + extensions with the program itself
    - CO-RE, notion of "only allow reads" / "only allow writes", that sort of "capability", optional / possibly future functionality
- Some discussion about moving relocations into the kernel
    - Sign a loader program + bytecode
- Discussion on embedding the signature into the bytecode vs. attribute to the syscall
    - Andrii concern: Avoid changing ABI of the bytecode. Syscall API change is easier
    - Daniel: Partial signature verifications against multiple programs loaded at the same time?
    - Alexei: New instructions is not great.
- Verify signature during the load time, but then the program can be attached to various hooks.
    - kprobe, uprobe, fentry.
    - Security concern around a program being appropriate at one hook but not at other hooks.
    - This could be solved somehow through capabilities
- Opaque blob of signature that is passed through the verifier to the signature checker program. Avoid kernel needing to know the specific format.
- Discussion on signatory service impact on availability during a serious incident ("cluster is on fire, I don't want to rely on the network etc. to debug and get it sorted").
- John: Overall this discussion is solving one minor issue but there's still major gaps elsewhere.
    - Can still write random maps and change the behaviour of the system with these signed BPF programs. Could be even bigger security issue than all of this.
    - KP: This solves a supply-chain issue. Avoid someone sneaking something in.
- KP: Important to support dynamic use cases. If we can trust the program that loads the BPF programs, then we trust the programs that it loads.

xattrs

KP

kfunc

Yonghong Song
11:00 Tuesday

Debugging OOMs (MM)

Kent Overstreet
11:00 Tuesday

- Kernel OOM reports kinda suck
- Exposing shrinkers - important for facilitating memory reclaim
- Mem report hasn't changed in 15 years. Last touch was reorg.
- Pretty printers unknown, "%p" - no cscope links, not discoverable
    - Patch: explicitly pass pretty printer function i the snprintf()
    - Make the kernel logs prettier
- LRU vs. SLAB check - proportions. Can indicate userland problem vs. kernel
- Some discussion about ratelimiting OOM / memory pressure reports
    - If it gets too expensive to print the report, it's easy to defeat a ratelimiter.
- Memory usage by type hierarchy
- How does someone allocate memory without accounting it?
    - Two calls, allocate and attribute it separately.
    - Can we switch it? Lots of callsites
- Granular pr_debug
- Use tracing hooks
- BPF to track maps by callsite
    - Outstanding memory allocations
    - Some concern about deallocation paths getting interrupted & losing events
    - Steven: "Synthetic events"?
- Broad concern: Looking for alloc + no free, if we are using memory to track all allocations then these must be tracked effectively forever. Too much overhead.
- Kernel log are not kernel ABI!
    - Google scripts parsing this will break.
    - This is OK! the OOM report should be useful for users

Making BPF programs more pleasant to write

Yonghong Song, Meta
11:30 Tuesday

- Parallel loops
    - John: Difficult, tried with `intrinsics` before - pass these through from the source program through to the BPF assembly to provide to verifier. How to get the verifier to figure out the flow without a full control flow graph (CFG).
- Compiler macros / attribute
    - Attribute to identify when a struct will be redeclared, so the latter declarations can be ignored.
    - Redefinition of C macros?
- It's possible to compile C++ to BPF

BPF CI

Mykola L, Meta
13:00 Tuesday

- Patch series run here: https://github.com/kernel-patches/bpf/actions
- Discussion about whether some of Cilium's nightly tests could/should move into BPF CI?

Clang regression checking

John Fastabend, Isovalent
13:30 Tuesday

- Latest clang broke some BPF loads because Linux verifier doesn't know how to interpret the latest LLVM internal optimizations (non-BPF changes).
- How to avoid maintaining our own fork of Clang
- How to also validate against multi-platform (different verifiers)
- Improve detection / build-bot, feedback loop to LLVM folks

BPF Observability tools update

Brendan Gregg, Intel
14:00

- Open issues:
    - Uprobes speedup
    - Heap tracing
    - LBR/frame-pointer/ORC walk merging
- Apparently Meta at the moment working on them

BPF Guidelines for newcomers

Brendan Gregg, Intel
14:30

- https://tinyurl.com/bpfguidelines2022

Wednesday May 4, 2022

Pod networking efficiency

Daniel Borkmann, Isovalent
09:15 Wednesday

- Themes: Retaining important metadata, sk
- Next step: delivery from host netns into pod with direct call, avoid backlog?
- uveth device driver:
    - Consensus that each peer dev will own a BPF program directly (can be of type sched_cls so tc BPF progs can be reused) - prog part of driver
    - uveth's dev_queue_xmit() switches skb->dev pointer to peer dev (hence netns switch), and right away calls into the BPF prog it owns. Prog can then redirect to phys dev in hostns, w/o going via per-CPU backlog queue. Or push up the local stack.
- Rebalancing connected UDP backends after socket LB
    - Don't put it in the fastpath as part of sendmsg.
    - Socket iterator for all netns'es + look at destination for the target backend (that's going away)
    - Still need a bpf_connect helper call from this new hook so that we can trigger the rebalance
    - Some discussion on a "socket kill" helper could also be useful in general.
    - Iterator + new connect/kill helper seems best way forward

Better algorithms & core improvements

Andrii Nakryiko, Meta
09:45 Wednesday

- Tries, RBTrees?
- Hashing algorithms - improve these, improve performance everywhere
    - Networking is moving from jhash to siphash (?)
- Ringbuffer improvements
    - User -> Kernel ringbuffer submits
    - Kernel code also submits events into BPF ringbuffer?

Overlayfs for cgroupfs+bpffs

Hao Lu, Amir G
10:20 Wednesday