

One Layer of the Onion: A Daemonless eBPF LSM for Confidential VMs in a WhatsApp TEE

Joshua Lilly
jgl@meta.com

Liam Wisehart
liamwisehart@meta.com

<https://github.com/facebookincubator/bpfjailer>

Agenda

- 01 Problem Statement
- 02 BpfJailer
- 03 CVM and Private Compute Platform
- 04 Daemonless Install
- 05 Signed Binaries
- 06 Self Protection and Signed Upgrades
- 07 Challenges

BpfJailer

- Created to sandbox/jail untrusted workloads
- For Private Compute Platform (PCP) provides system-wide Mandatory Access Control
- Policy tied backed to cryptographic root of trust

A WIP complete rewrite using eBPF is now open source

- Lacks many features at this time, **has rough edges**, will eventually replace internal
- Daemonless, pinned programs/maps only, with utilities for updates
- Supports 6.16+, will (eventually) move to 7.4 to gain advantage of increased stack size
- Consolidates hundreds of maps into an arena heap, uses libarena for common utilities

OSS version is new, was shipped quickly, still being developed

<https://github.com/facebookincubator/bpfjailer>

Coverage

Closed Source

Open Source (WIP)

Networking

- TCP bind()
- TCP connect()
- UDP sendmsg()
- UDP recvmsg()
- Unix domain bind()
- Unix domain connect()
- Unix domain sendmsg()
- Vsock bind()
- Vsock connect()
- accept()
- DNS
- DBus
- TCP and UDP Proxy
- CA injection and local TLS interception
- Bpfilter integration (planned)

Execution

- Exec binary and arguments
- setuid/gid flags

Filesystem

- File path matching
- mount()
- ioctl()

IPC

- SysV
- Posix

Kernel

- Kernel modules
- eBpf programs and maps

Tampering

- ptrace()
- process_vm_readv()/writev()
- kill()
- /proc

Signed binaries

Signed upgrade flow

More to come!



BpfJailer Permissions

Processes placed in pods with defined roles

- Can be nested (ex. host -> container -> workload might have different roles)
- Different resources can be owned by a pod (ipc objects, bpf maps, keyrings, processes, etc.)
- Pods have different permission sets, can be structured to further reduce or grant permissions

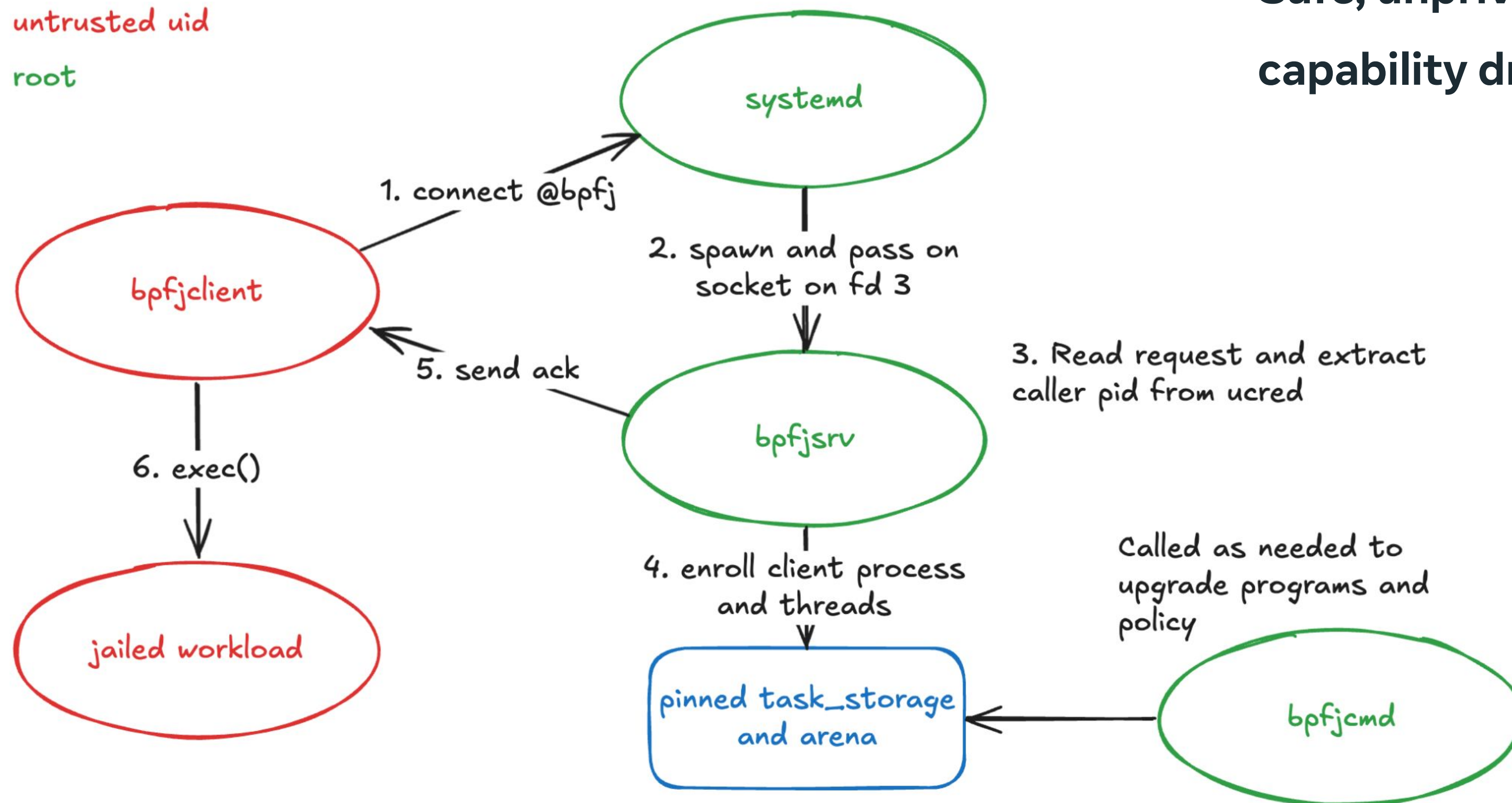
Pods can have a set of variables (dynamic policy item set during pod enrollment) to isolate resources

- For example a container with variable \$ID and various inner jails with \$VM_ID
 - Filesystem: `/run/containers/fs- $\{ID\}$ /root = RDWR`
 - Unix socket: `/run/containers/fs- $\{ID\}$ /root/run/sock/vsock- $\{VM\_ID\}$  = ALLOWED`
 - POSIX message queue: `/mq- $\{ID\}$ -* = ALLOWED`

Path matching done in the init mount namespace to prevent manipulation like namespaced
bind mounts

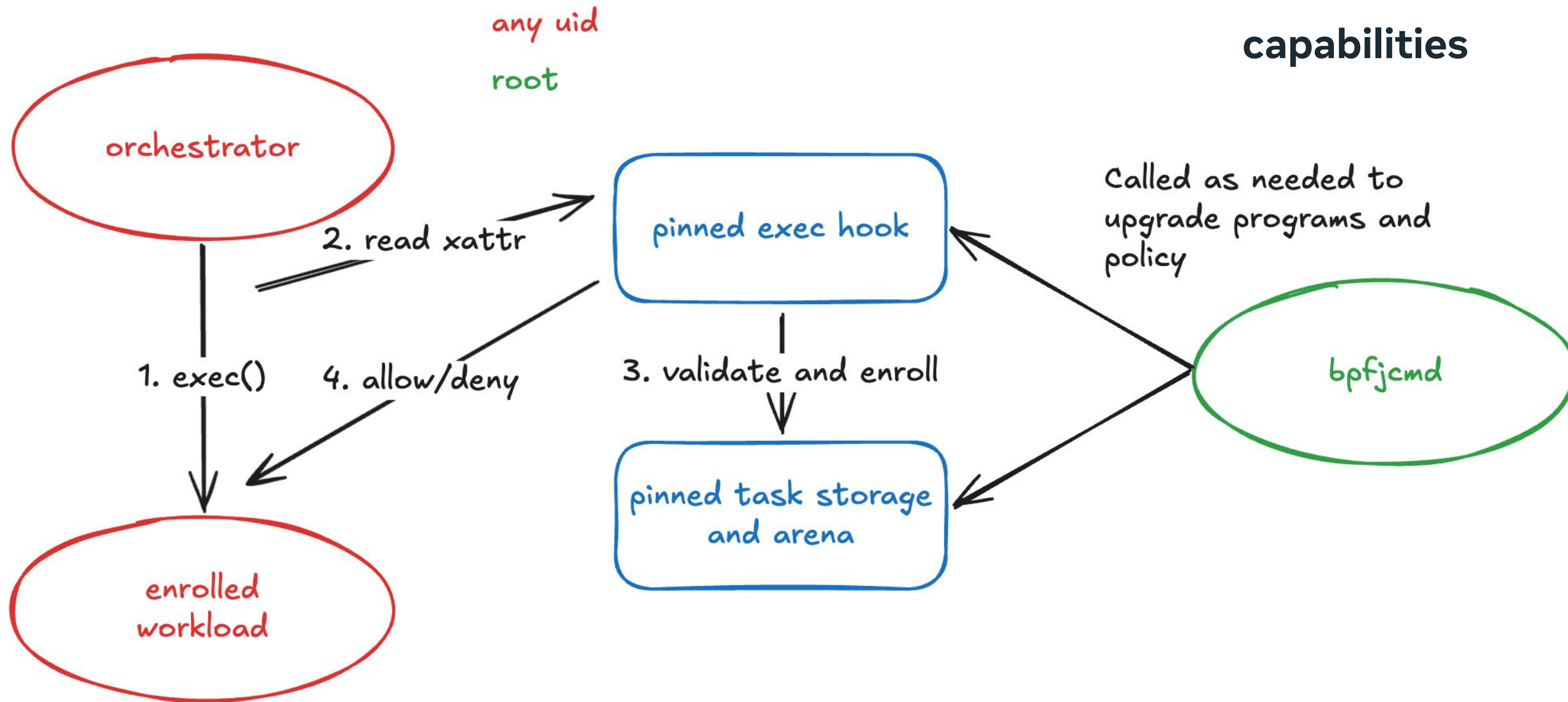
BpfJailer Socket Enrollment

Safe, unprivileged
capability drop

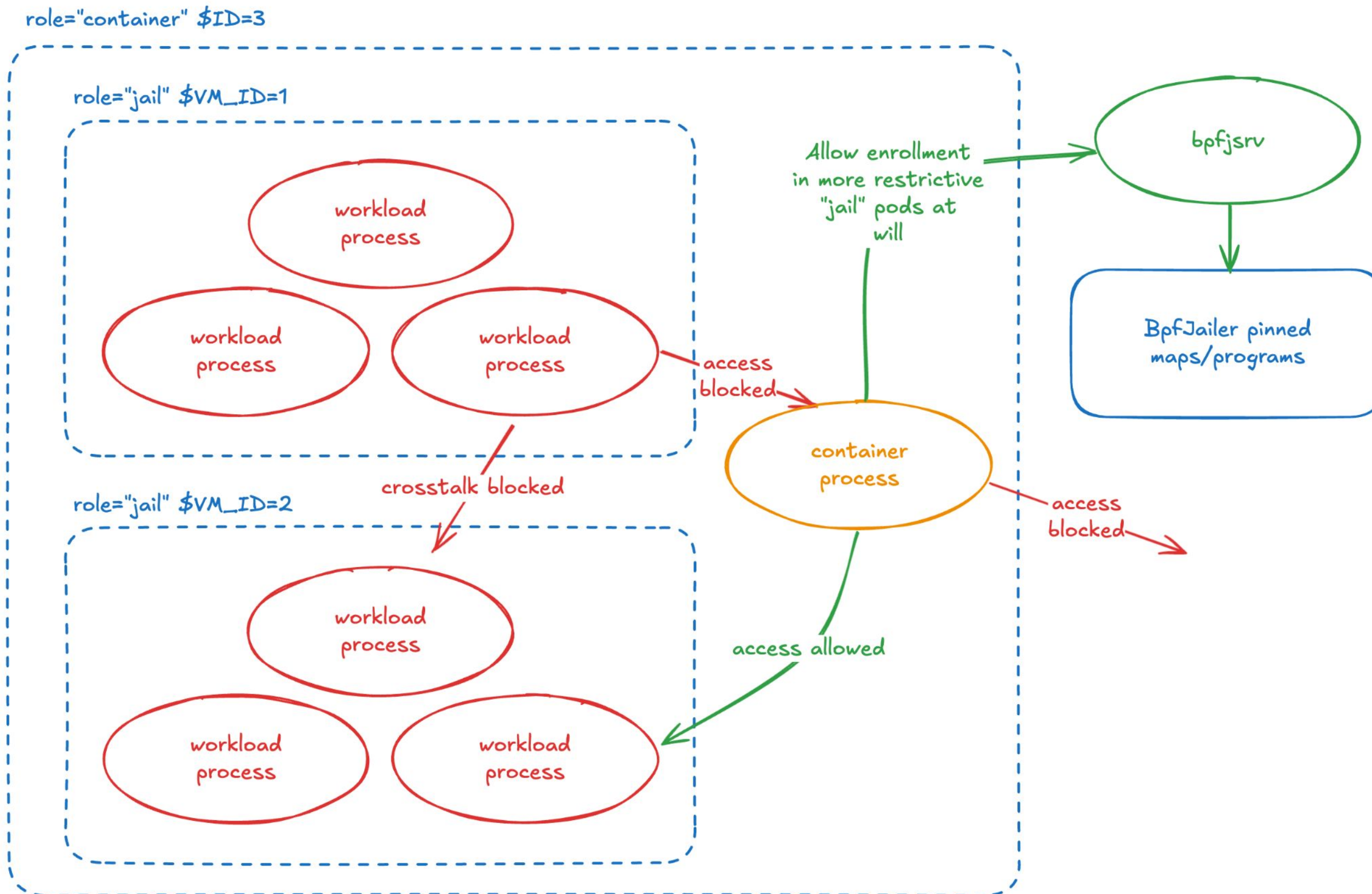


BpfJailer Xattr Enrollment

With signed binaries, can
allow safe acquisition of
capabilities



BpfJailer Data Model



One way to set things up:

Bind mount bpfj socket into container (first level jail) so it is accessible

Only allow self enrollment in more restrictive role (can only drop caps) to create more restrictive jails, policy is a union of both roles

Signed binaries can gate enrollment if caps need to be raised

Meta as an Adversary

- AMD SEV-SNP provides encrypted memory and protection from the hypervisor
- Exploits have been found, so far all have been patched
- An insider or outsider with root access could potentially use an unknown zero-day to attach the enclave

We set a bar of a user's data being confidential, no matter who, or what access an attacker might have

Binaries and policy on the system must be tied back to CVM attestation measurements and transparency ledger

<https://ai.meta.com/static-resource/private-processing-technical-whitepaper>

Daemonless by Design

- A one shot binary in init attaches and pins programs to the bpffs and loads all policy and certs
- bpffs in /sys, survives init, no processes to worry about
- No resident process to kill, OOM, or exploit
- Logger is managed by systemd post-init and is not trusted (only has access to logging ring buffers)

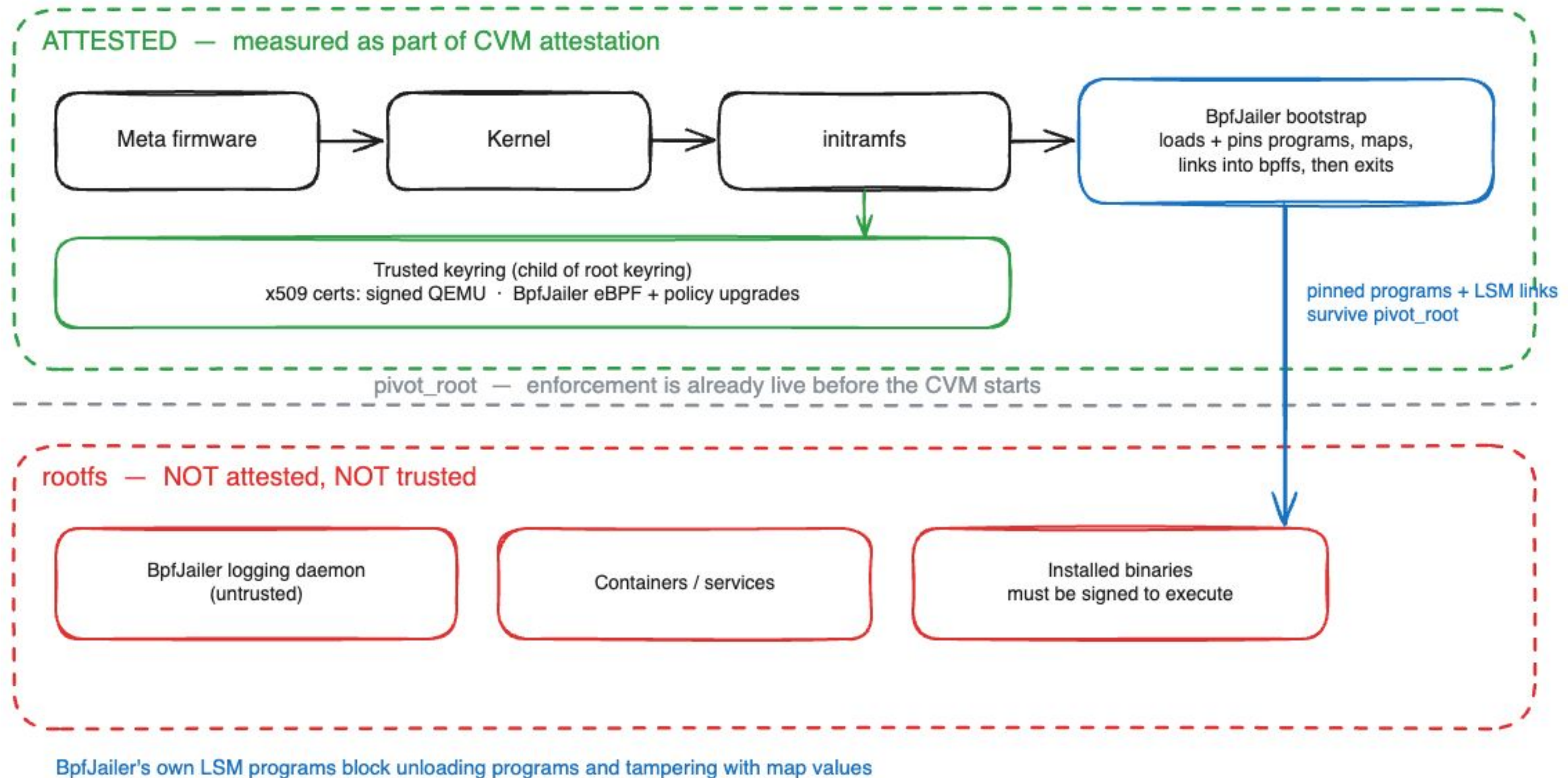
Trust extension from init to rootfs

- At Meta firmware, kernel, and initramfs are attested and part of CVM attestation report
 - By including BpfJailer in initramfs it is part of that attestation measurement
 - Extends that trust boundary selectively to the more chaotic rootfs/chef/tupperware managed layer through signed binaries and other access control

Daemonless vs. Resident Daemon

Daemonless (open and closed source)	Resident daemon (closed)
Reload difficult, can result in a bricked system	Easy dynamic policy reload, daemon can just be killed to reset system
Low steady-state footprint, logging not secure	Have to pay for a daemon, rich userspace logging
Low attack surface	High attack surface, but easy to debug and observe
Survives pivot root	Must be installed in rootfs or later

Initramfs Install



Protecting BpfJailer From Removal

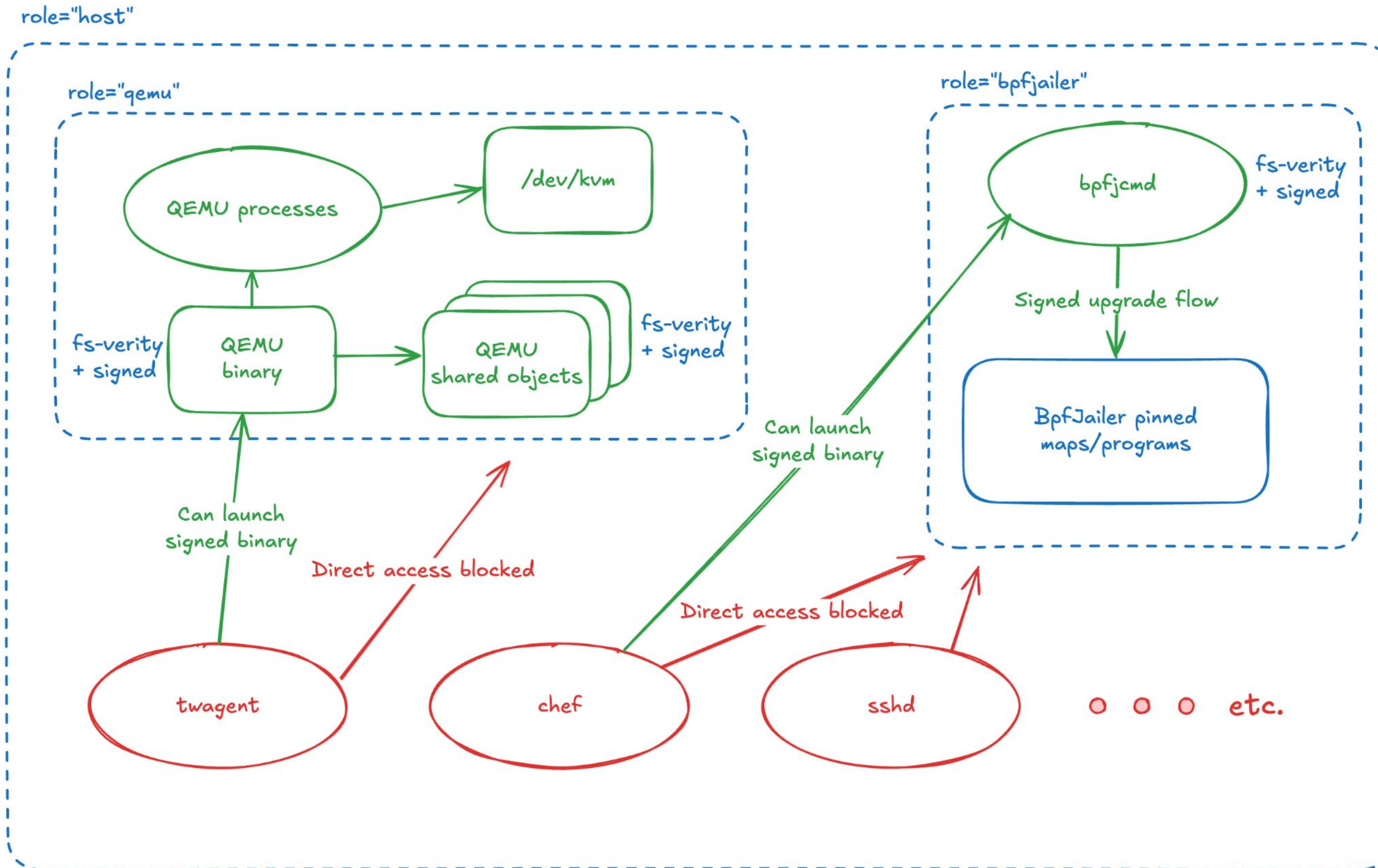
- Pinned programs and maps are protected by BpfJailer's own LSM programs
- bpfjctl invocations are verified by the currently running BpfJailer
- Every shared object, binary, and file must be signed by a trusted key
- The tension how do we killswitch under a static install
- Immutability is a security property and an operational hazard

Why This Makes Deployment Hard

- Early = powerful. Early = inflexible. Same property.
- Fleet policy change is a boot-artifact rollout, not a config push
- Any change is coupled to measurement, and therefore to attestation
- Rollback isn't systemctl restart, it's a reboot
- Kernel and kfunc dependencies must be validated against the exact boot image

Private Compute Platform

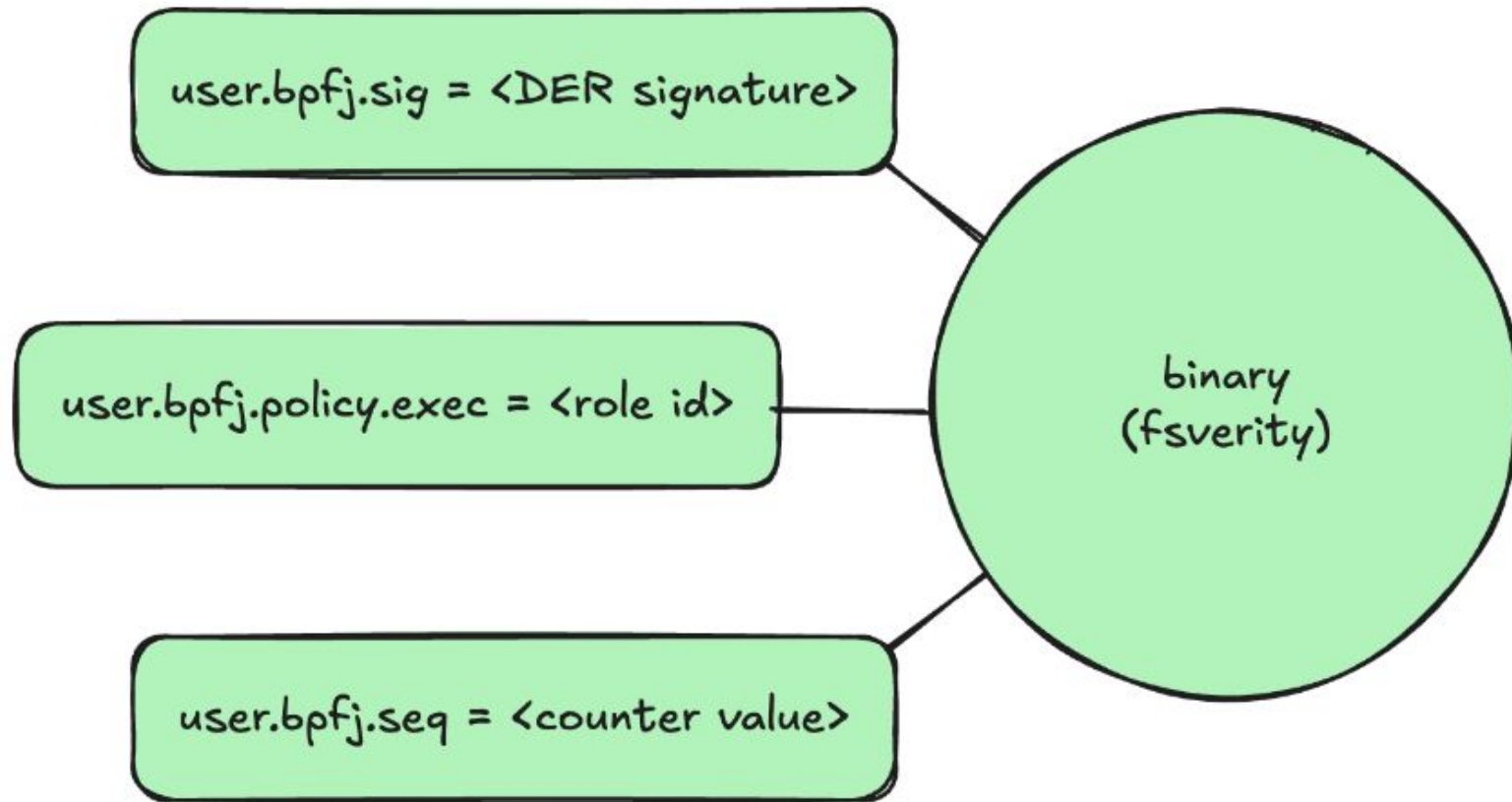
Need to protect CVM and QEMU from tampering



- Signed binaries
- Ptrace and /proc blocked
- Key files protected
 - /dev/kvm
 - /sys/fs/bpf/bpfj

Rules installed by initrd and immutable + measured as part of attestation

Signed Binaries



What is signed



Verified against a set of certs in keyring associated with a role id

- role is verified by cert used in signature
- sequence number is signed and prevents rollback

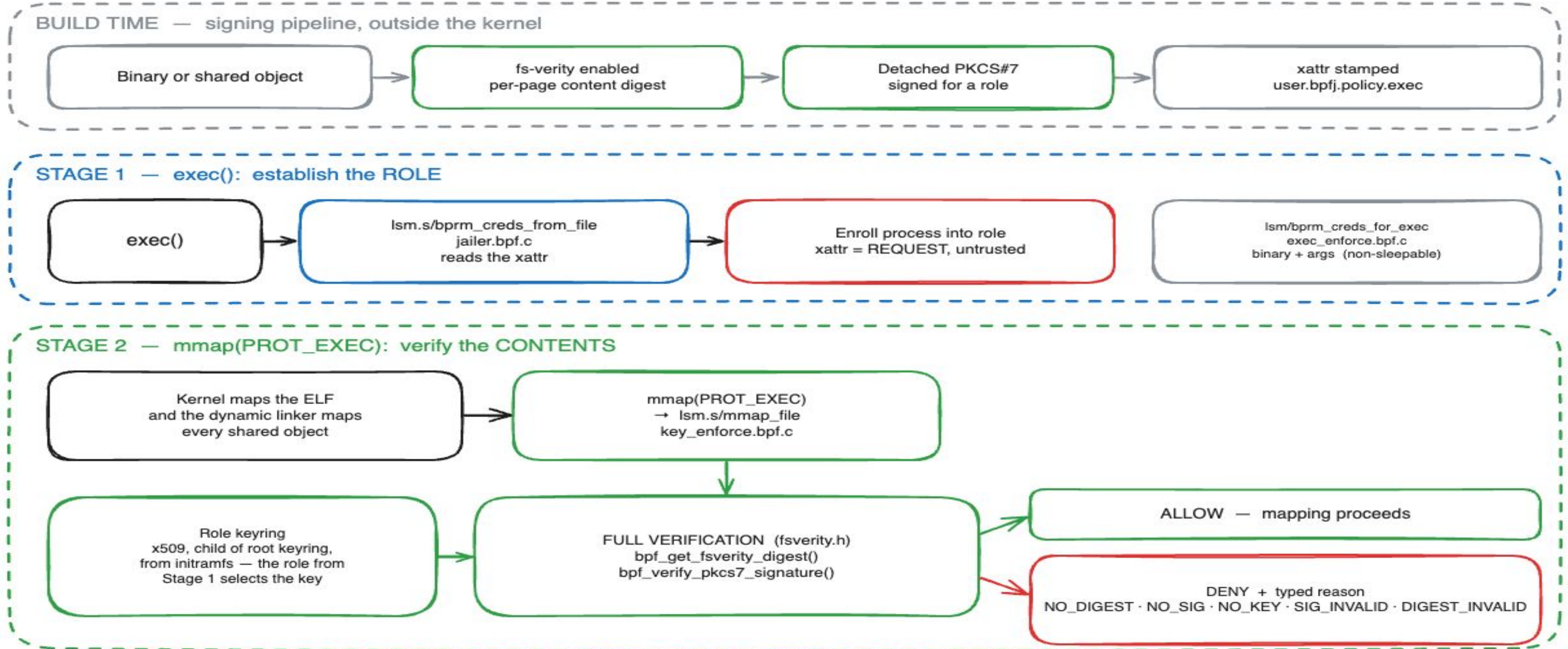
Currently all arguments, env variables, and policy files baked into single static binary in bpfjcmd

We will also support argument validation and/or signing, signing a separate policy file, etc.

Also check execed binary + mmaped executable files (for dlls)

Variables not supported for xattr enrollment at this time

Signed Binaries Verification Flow



Every PROT_EXEC mapping is verified — the main binary AND every shared object, including dlopen(). exec picks the role; the mapping is what gates the bytes. No path trust: identity is content + signature.

What Verified Identity Buys at Runtime

- QEMU execs, gets verified, and is pinned to a verified role
- Only signed QEMU reaches /dev/kvm and /dev/sev
- ptrace_access_check and ptrace_traceme stop any debugger attaching to the guest process
- task_kill stops anything on the host signalling it into a fault
- /proc and process_vm_readv/writev closed to protected processes

Need to also validate env variables, command line arguments, and any important files being opened

Limitations

- Not a defense against a compromised kernel — signed kernel modules + attestation
- Not a defense against pre-boot or firmware compromise — remote attestation and secure boot
- Not a defense against guest-internal compromise
- Not a defense against a signed-but-malicious binary
- Not a substitute for provisioning controls on what reaches disk
- Enforcement can also brick the host it protects

Still a few eBPF hacks required, but the situation is massively improved with arena data structures

Upgrade Flow

- Initially used a static version of BpfJailer
 - Upgrades require reboot
 - Killswitch is impossible
- Moving to signed upgrade flow
 - Currently running BpfJailer version cryptographically verifies new version of BpfJailer
 - Policy or other instructions are verified
- Working on supporting certificate expiry, rotation
 - eBPF signature helper bases expiry time on time of signing, not time of validation
 - Need a trusted clock setting option (might be included in BpfJailer updates)

bpjctl and bpjcmd

- Command-line tool for loading BpfJailer programs, maps, and policy
 - bpjcmd bakes a single operation (including policy) into a static, signed binary for easy validation
- Can enable, disable, update keyrings, etc.
- All invocations verified by the currently running BpfJailer and blocked otherwise with LSM
- All shared objects, binaries, and files need to be signed by a key trusted by BpfJailer in order to allow the operation

Logging: UDP from eBPF, Now Upstream

Mahé Tardy (Isovalent / Cisco)

Problem:

- Difficult to keep logger daemon in sync with the pinned program versions
- Logger daemon is much easier to attack and bring down than the pinned programs
- Easy to tamper with the network logging

Solution:

- Log emission directly from BPF, over UDP
- eBPF already has needed encryption primitives if desired
- No userspace consumer required
- Works pre-pivot_root, before any rootfs exists
- Survives the death of any daemon

One Layer of the Onion: A Daemonless eBPF LSM for Confidential VMs in a WhatsApp TEE

Joshua Lilly
jgl@meta.com

Liam Wisehart
liamwisehart@meta.com

<https://github.com/facebookincubator/bpfjailer>