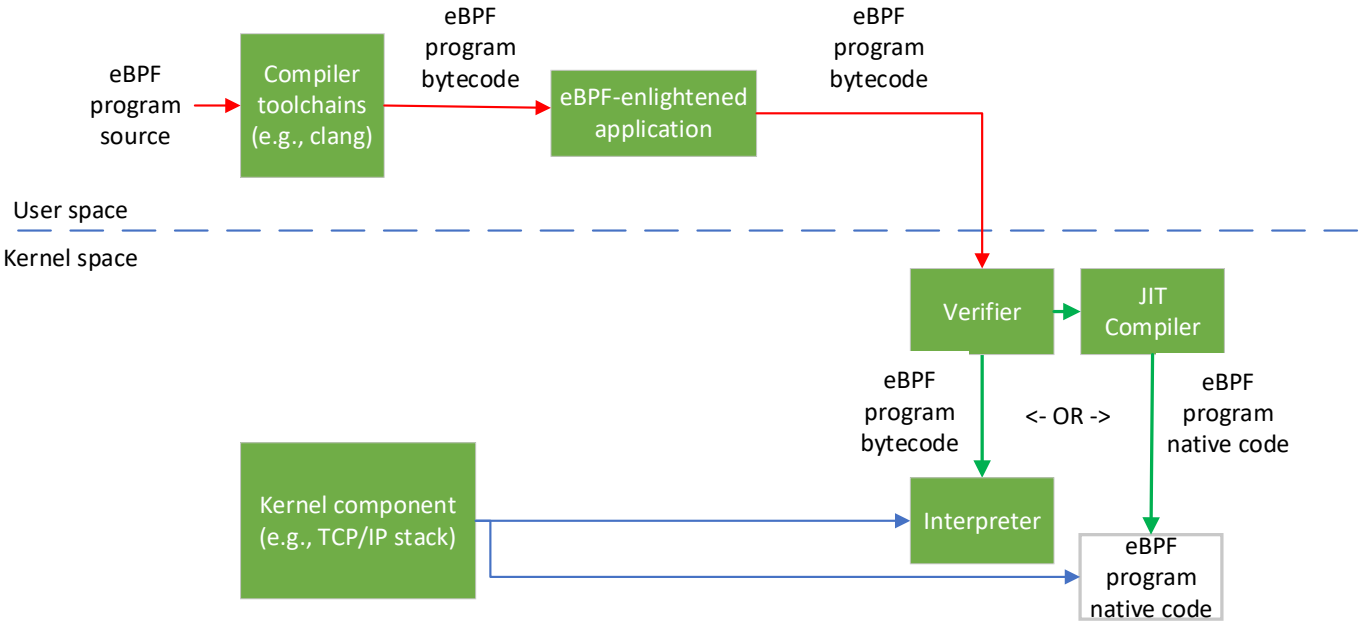
Signed eBPF Programs

A Cross-Platform Analysis

Dave Thaler

@ThalerDave

eBPF



Code is permitted if program is provably safe **AND source is authorized**

- Various definitions of “source is authorized”, which is where code signing comes in
- In some cases, code signing could also cover “provably safe”, allowing offline verification

Classic Code Authorization Methods

	(Type 1) HyperVisor-enforced Code Integrity (HVCI)	Binary Signing	Volume Signing
Granularity	Code page	Binary	Entire volume (per block)
Enforced by	Hypervisor	OS	OS
Dynamic code allowed	No	Yes	Yes
Example	Microsoft Hypervisor	fs-verity	dm-verity

"Static" functionality is where all code is fixed at design/compile time
"Dynamic" functionality is where details not known until runtime (e.g., dtrace, bpftrace)

- "Dynamic code allowed = No" means signing would be required at runtime
- But code signing keys are usu. offline, accessed via some build/signing pipeline

Bottom line: HVCI is (intentionally) incompatible with scenarios requiring dynamic code

What form of eBPF program is signed?

Source code: Any popular language; C, Go, Rust, …

Byte code: eBPF instruction set used by verifier, JIT compiler, ~~interpreter~~. Often stored in ELF format.

Position-independent code: Processor-specific (e.g., x64) code, pre-relocation. E.g., .sys file.

Native code: Result of relocation, symbolic refs replaced with usable addresses specific to the machine. This is what actually executes in memory.

---

eBPF-enlightened application: user-mode app that loads/communicates with eBPF program

Does signature cover anything else?

Program-only: signature covers single binary (byte code, PIC, or native code)
- HVCI requires signed PIC

Package/Container: signature covers a user space app + any eBPF programs it uses
- Can be distributed through an app store, Kubernetes, etc.
- L3AF uses packages, bumblebee uses containers

Repository: signature covers entire volume, directory, or other repository
- Authorization just checks whether eBPF program came from such a repository

Source of program: eBPF program itself is not signed, only the app generating/loading it
- Supports dynamic code without online signing key
- Requires interpreter (side channel attack concerns) or dynamic code page support (no HVCI)

With this diverse set of signing scenarios, how many different solutions do we need?

Introducing the “Gatekeeper” concept

(credit: John Fastabend)

Flexibility is essential, not hard-coding the answer

- Example #1: HVCI vs bpftrace
- Example #2: policy by form or what signature covers
- Example #3:
  - Typical Windows policy = only Microsoft signed code can be loaded in kernel, since Microsoft responsible for support
  - Typical Linux community policy = distro should not control what programs can run; admin is in control
- Example #4: custom policies based on which keys are trusted, execution mode (JIT/interpreted/etc.), user ID, location, etc.

Enter the **Gatekeeper**: programming the policy using a special eBPF program

- Runs on-device to meet availability & latency requirements for eBPF use cases

Gatekeeper details

Absence of a gatekeeper loaded means fall back to hard coded default

Strict default (e.g., only permit eBPF programs signed by the distro vendor, like Windows):

- Gatekeeper can be loaded any time after startup, to loosen the policy

Loose default (e.g., permit any program submitted by an admin):

- Gatekeeper must be loaded before anything it is intended to deny
- **Linux (eBPF is part of kernel itself):** load as part of boot process
- **Windows (eBPF is separate driver):** load as part of starting eBPF driver

Gatekeeper is still applicable in both cases

Key Takeaways

1. Signing programs (or their sources) can allow increased security
2. Signing and gatekeeper concepts apply across multiple OS's and distros
3. Same architecture can accommodate multiple types of signing and distribution mechanisms
4. Allows custom policies without taking a position on the tussles
5. Tussles left for individual OS's, distros, and admins to deal with on top