ISOVALENT

BPF signing using fsverity and LSM gatekeeper

Lorenz Bauer (2023-05-10)

eBPF is very powerful

- With great versatility comes a potential for abuse
  - Exfiltration
  - Keylogging
  - etc.
- By necessity, this has to limit the versatility of bpf()

Privileged, unprivileged, signed, what?

Problem	Solution
Unprivileged bpf()	Turn it off
Give CAP_BPF to programs in user namespaces (Meta)	BPF token (Andrii Nakryiko)
Allow untrusted, unprivileged programs to load specific BPF (Google)	Sign the BPF bytecode (KP Singh)
All executable code must have an authorized source (Microsoft)	Establish authorization (Dave Thaler)

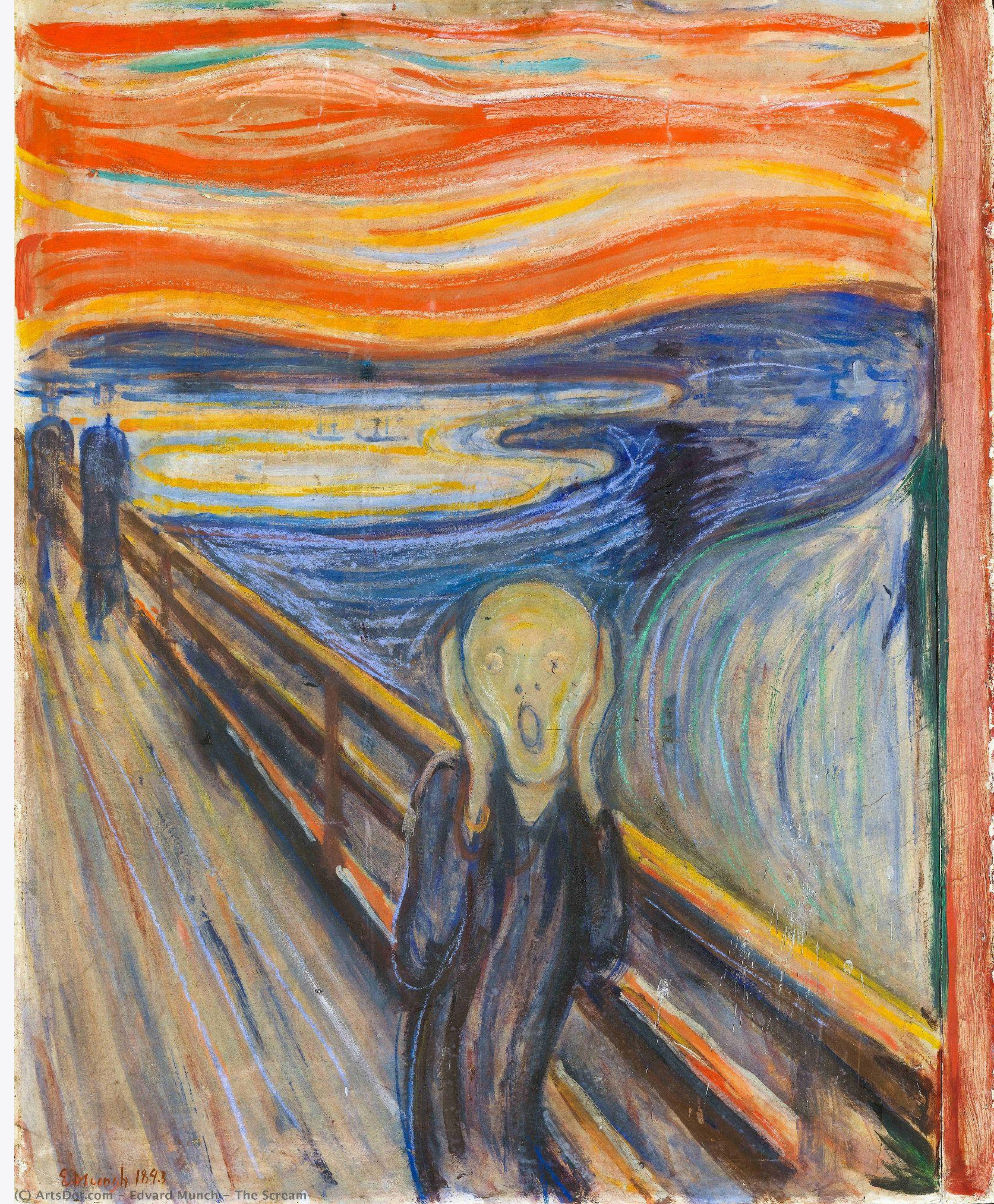
Sign the BPF bytecode

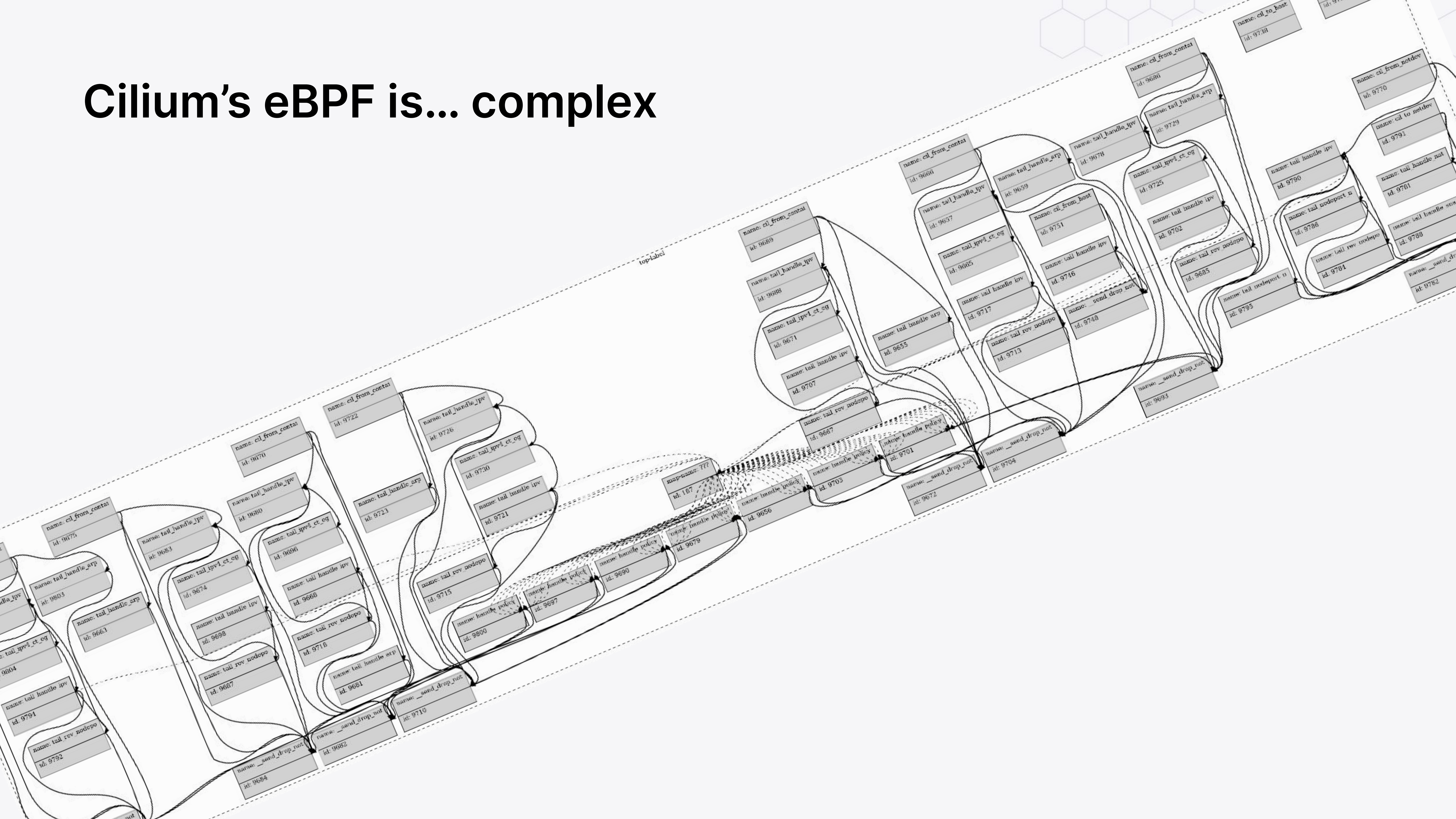
```
sign(hash(struct bpf_insn insns[] = {...})
```

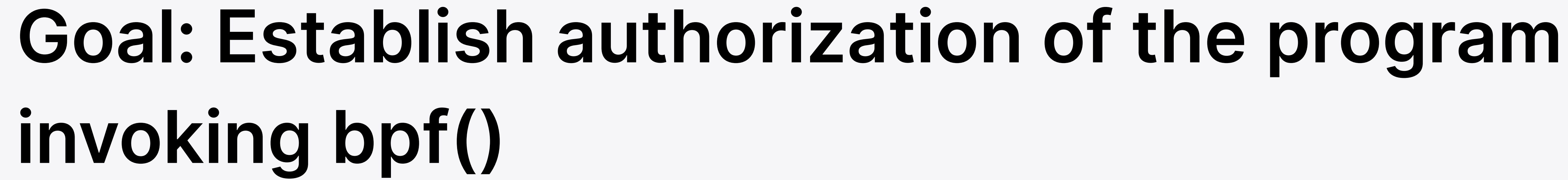
This is not crypto advice!

Sign the BPF bytecode

A Cilium developer's reaction to bytecode signing.




Cilium’s eBPF is... complex



Goal: Establish authorization of the program invoking bpf()

- Bless the binary that accesses bpf(): bpftrace, cilium, etc.
- Orthogonal to signed byte code: both could be active at the same time

PoC: What do we need?

1. A way to identify a binary (hash) and protect it from modification
2. A way to express trust in an identity (signature)
3. A way to express a policy

1. Identify a binary and protect it from modification

- fsverity: fast per file integrity mechanism
- Easy to enable on a single file:

```
fsverity enable /path/to/file
```

2. A way to express trust in an identity

- IMA has a lot of the things in place I need
  - Signature format and xattr storage
  - In-kernel caching of metadata, verification results
  - User space tooling to fiddle with things
  - Key management*
- Some interesting integrations
  - rpm support
  - Keylime remote attestation

3. A way to express a policy

- BPF LSM + new kfuncs

DEMO TIME

Let's hope this works

https://github.com/isovalent/bpf-verity

```c
int BPF_PROG(bpf, int cmd, union bpf_attr *attr, unsigned int size) {
 struct task_struct *current = bpf_get_current_task_btf();
 struct file *exe = get_task_exe_file(current);

 if (!exe)
   return 0;

 int ret = bpf_ima_file_appraise(exe);
 fput(exe);

 if (ret == -ENOENT || ret == 0)
   return 0;

 return -EPERM;
}
```

Takeaways

- It works!
- “Identity” should be pluggable: fsverity, dm-verity, ...
- Signatures should be compatible with existing tooling
- “Trust” should be flexible

Does the IMA trust model work for us?

- .ima keychain contains trust anchors
  - Adding to the keychain can be restricted to signed public keys
    - Root of trust is the system keyring
    - Extend system keyring via Machine Owner Key (of Secure Boot fame)?
- Need a way to express “I only want a subset of signed programs to have access”
  - Additional keychains?
  - Store root of trust in BPF token?

How much do we integrate with IMA policy?

- Currently require an appropriate IMA policy in place to populate signature metadata
  - Make BPF kfuncs measure on-demand?
- BPF LSM is like a dynamic “appraise” rule, how should this be logged / audited?
  - Make BPF appraisal a first class concept in IMA?

ISOVALENT

Thank you!

lmb@isovalent.com

Alternatives to IMA signatures

- fs-verity signatures (FS_VERITY_BUILTIN_SIGNATURES)
  - Basically kernel module signatures (PKCS#7)
  - Considered a proof of concept by the author
- “Cloud Native Signatures”
  - cosign / notary: sign OCI bundles (aka fancy gzip)
  - Not clear how to bridge to file / filesystem image
- custom “BPF signatures”
  - We can verify PKCS#7 from BPF today
  - Lots of plumbing to be done, not very exciting and hard to get right
  - Doesn’t integrate with the rest of the kernel infrastructure

fs-verity alternatives

- fs-verity needs filesystem support
  - currently only ext4, f2fs, btrfs
- Could use IMA digest or dm-verity instead

Design Choices ⇨ Tools ⇩	device vs file	kernel vs user	writeable?	chunks?	tree vs. list	lazy or eager?
openssl dgst	file	userspace	N	whole	N/A	eager
rpmsign	file	userspace	N	whole	N/A	eager
Integrity Measurement Architecture (IMA)	file	kernel	N	whole	N/A	eager
fs-verity	file	kernel	N	chunks	tree	lazy
dm-verity	device	kernel	N	chunks	tree	lazy
dm-integrity	device	kernel	Y	chunks	list	lazy
btrfs HMAC	file	kernel	Y	chunks	list	lazy

Via fs-verity support in btrfs (Meta)