BPF Signing + IMA
What do we want?

"This BPF program comes from a trusted source"
Requirements

Flexibility

Stable BPF instruction buffer
About IMA

What?
A policy framework that allows users (e.g. distributions) to guarantee the integrity of the system

Why?
Ensure the instructions / code executed on the system come from a trusted source
Key discussion points

How is the policy for signature verification specified?

How is the policy verified?
Policy specification: Should it be just IMA?

IMA should not be the only way the policy is specified

Not a flexible solution
Policy specification: Only custom policy formats?

No, some distros are already used to IMA and want to use it for BPF too.
Proposal #1

Support IMA policy format and also provide support for custom policy formats.
Verification Logic: Only in IMA?

Not everyone enables IMA, not a flexible approach
Verification Logic: Only in BPF?

- BPF exports helpers for IMA functionality
  IMA may not be willing to expose a lot of internals to eBPF

- Distro implements (or uses) an eBPF program that understands the IMA policy
  Maintainership of the IMA signature verifier eBPF program
Both IMA and BPF programs should be able to implement the signature verification
Proposal #3

All verification happens in the implementation of `security_bpf_prog_alloc`.

The LSM framework allows for co-existence.
How do we make it flexible?

The signature is stored in a buffer passed along with BPF syscall.

Anonymous blob or buffer, IMA can choose what to write.
union bpf_attr {
    [...] 
    struct { /* anonymous struct used by BPF_PROG_LOAD command */
        __u32 prog_type; /* one of enum bpf_prog_type */
        __u32 insn_cnt;
        __aligned_u64 insns;
        [...] 
        __aligned_u64 core_relos;
        __u32 core_relo_rec_size;
        + __aligned_u64 signature;
        + __u32 signature_size;
    };
};
IMA, an LSM?

```c
int security_bpf_prog_alloc(struct bpf_prog_aux *aux)
{
    int ret;

    ret = call_int_hook(bpf_prog_alloc_security, 0, aux);
    if (ret)
        return ret;
    return ima_bpf_prog_alloc(aux);
}
```

Some historical context..does it really matter? Doesn't override LSMs
Light skeletons

A stable instruction buffer is required for signature verification

Use light skeletons

Light skeletons are limited to only a subset of programs
Basic bpftool support

```c
int bpf_data_sign(
    const char *private_key_path,
    const char *x509_cert_path,
    const void *data, size_t data,
    void *sig_buf,
    size_t max_sig_len)
```

Can be extended to other formats e.g. IMA
Signing: PKCS#7

bpftool prog load -L -S signing_key.pem signing_key.x509 prog.o

bpftool gen skeleton -L -S signing_key.pem signing_key.x509 prog.o
Thank you!