**BPF Signing + IMA** 



#### "This BPF program comes from a trusted source"



## Flexibility

#### **Stable BPF instruction buffer**



#### 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.



# 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



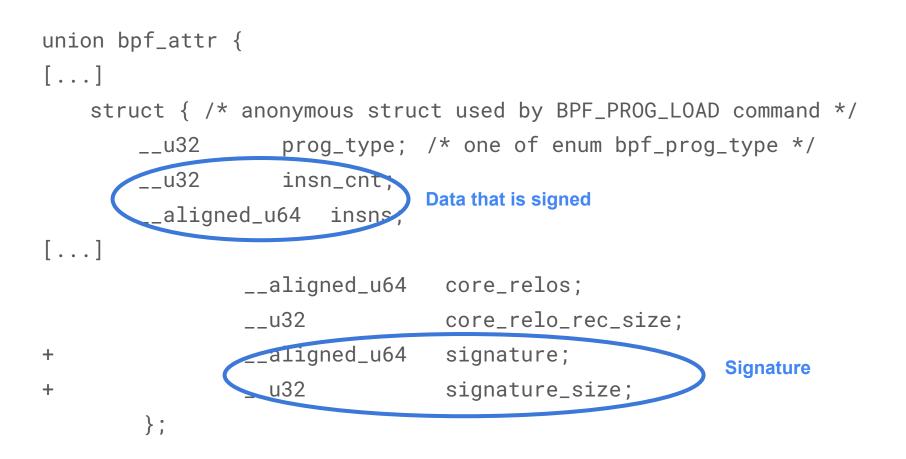
# All verification happens in the implementation of security\_bpf\_prog\_alloc.

#### 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.



#### IMA, an LSM?

```
int security bpf prog alloc(struct bpf prog aux *aux)
{
       int ret;
       ret = call int hook(bpf prog alloc security, 0, aux);
       if (ret)
                                        Doesn't override
               return ret;
       return ima_bpf_prog_alloc(aux);
                                        LSMs
}
```

#### Some historical context..does it really matter?



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**

```
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

