Watching the Super Powers

John Fastabend, Isovalent
Linux Plumbers Conference 2021

Alternative Titles

Title 1: BPF signing broke my tooling

Title 2: An argument for BPF runtime policy

Title 3: Its Bee’s all the way down

Agenda

0/ Signing Proposal
1/ Alice, Bob, and Eve
2/ Use Cases
3/ Alternate Proposal
4/ Summary

About Me

Kernel developer
BPF kernel developer
BPF user
Cilium developer
Isovalent Engineer

Kernel modules vs BPF programs

- Safety depends on diligence of developer:
  - Never terminate, memleak, use after free, etc.
- Built and distributed normally
- Built against kernel header files
- Typically standard kernel APIs netlink, proc, etc.
- Expectation of stable APIs
- Lifetime of modules years

- Safety built into loading process
  - Must terminate, memory checked, etc.
- **Often dynamically built and optimized**
- CO-RE patched at load time
- User/BPF interface “maps” “*ring” and “mmap”
- BPF developer can define lifetime of their API
- Lifetime of BPF program may be anything from stable product with years of support or just a single debugging session

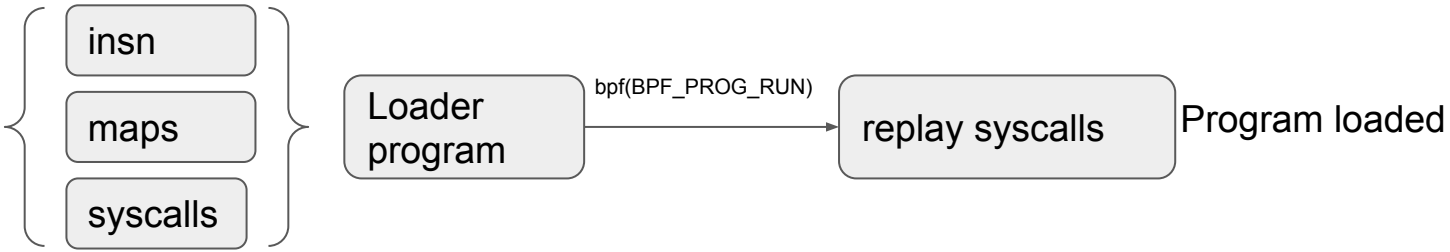
Signing BPF Programs (A sketch)

Requirements:

- signed object must be stable.
  . CO-RE pushed to late post signature verification
  . Map fd rewriting post to late post signature verification

- ‘Loading’ a BPF program is a multi-step process
  . signature must capture the load process

- Signature verification captures loader program including, load steps and user code

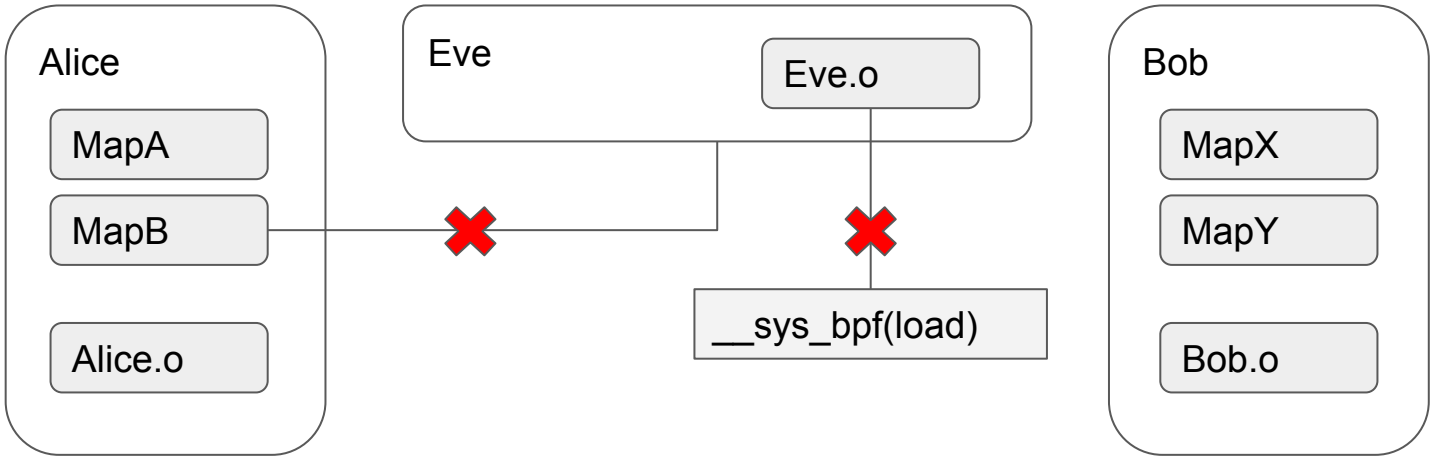


https://lore.kernel.org/bpf/20210514003623.28033-1-alexei.starovoitov@gmail.com/

BPF Signing does lockdown malicious Eve

- Given two Signed BPF Users: Program Alice and Program Bob
- And a malicious unsigned actor Eve

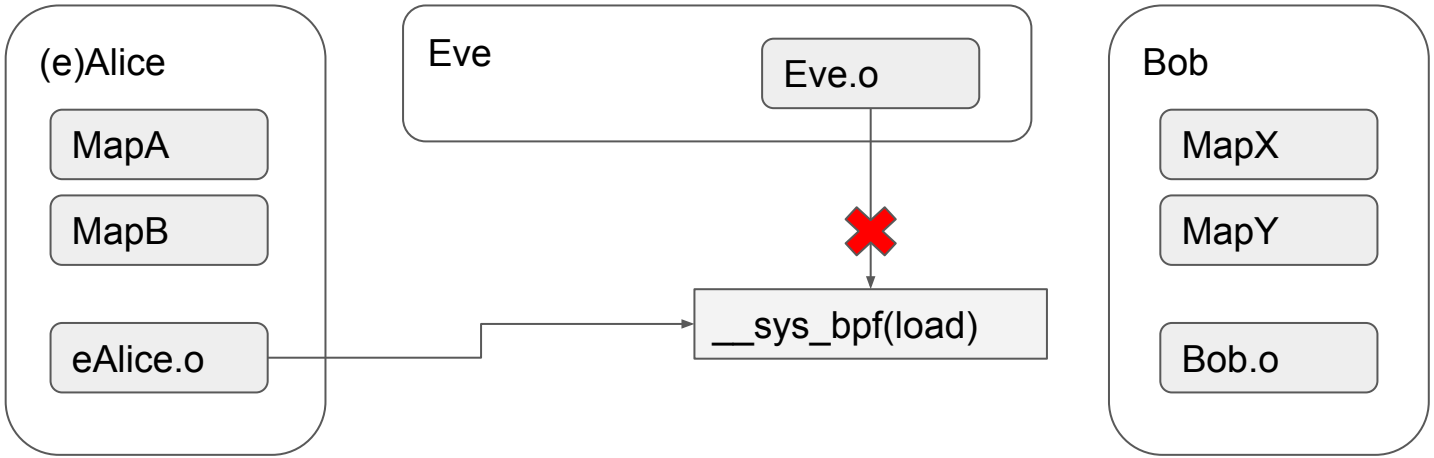
- Eve can not load programs
- Eve can not read/update/modify Alice's or Bob's map given correct file system



BPF Signing does lockdown Imposter Alice

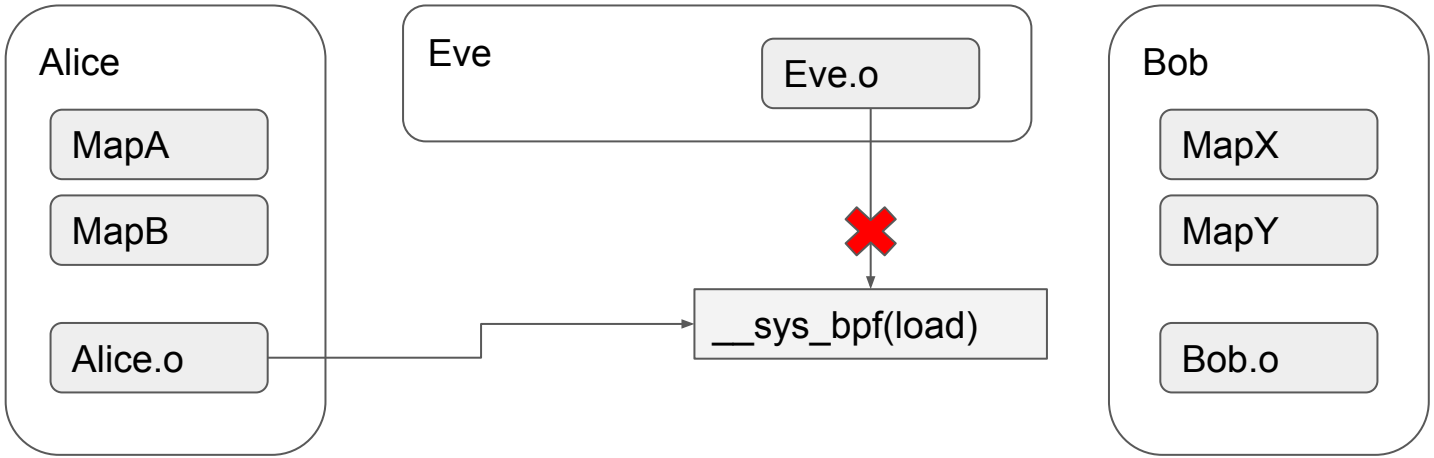
- Given two Signed BPF Users: Program Alice and Program Bob
- And an imposter Imposter Alice reporting to be Alice

- Alice can not load arbitrary programs



BPF Signing does lockdown Dynamic Alice

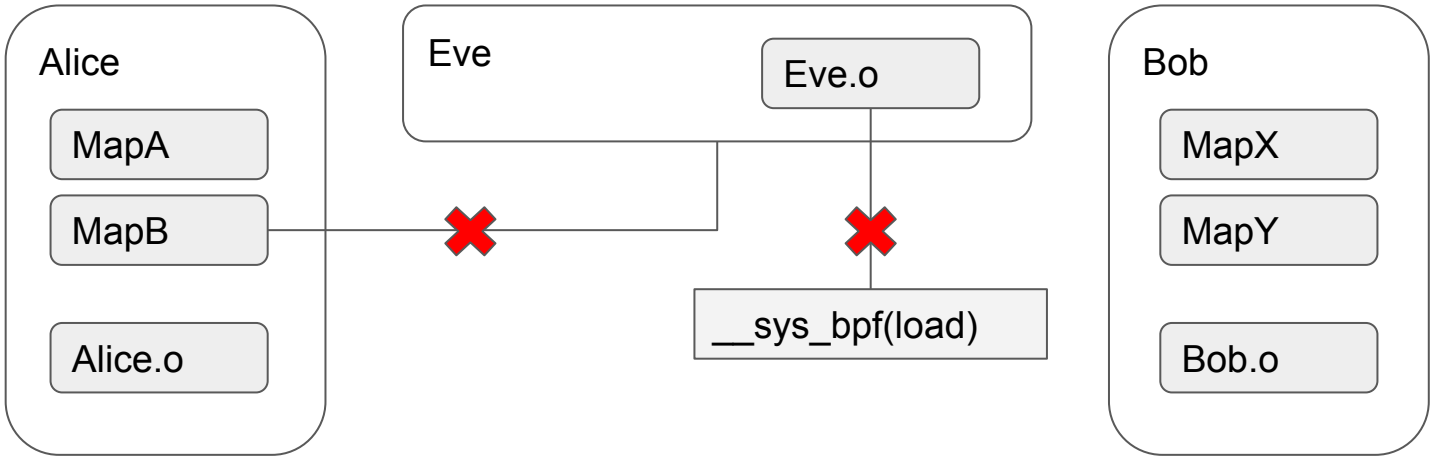
- Given two Signed BPF Users: Program Alice and Program Bob
- Alice implements a dynamic BPF program
  - tcpdump style filter
  - runtime generated networking program
  - BCC tracing hooks
- Signing will block ‘dynamic’ Alice at this point



BPF Signing: Cost

A malicious Alice and a good Alice using optimized or dynamic programs are not distinguishable and both are blocked.

COST of SIGNING: Dynamic code generation and optimizations are not supported breaking many existing tools and innovative tools yet to be imagined.



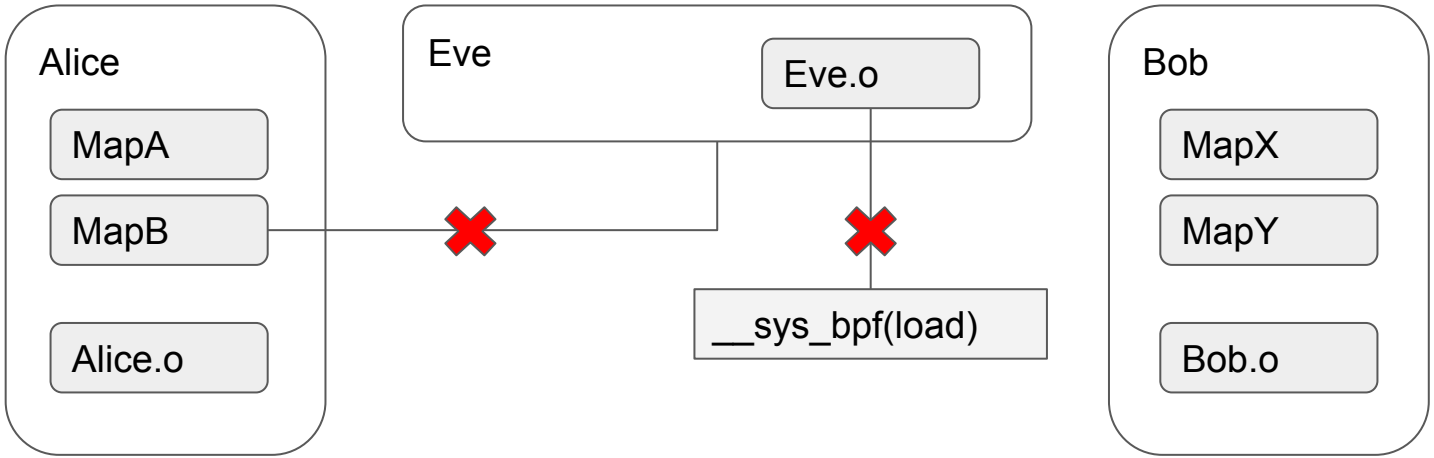
BPF Signing Breaks Use Cases

- Many BPF programs optimize and generate BPF codes on the fly
  - BPFTrace: High level language for generating observability tooling
    https://github.com/iovisor/bpftrace/
  - P4: High level DSL to generate networking datapaths, has a BPF backend
    https://p4.org
  - Cilium: Optimizing BPF code generates pod specific programs attached at runtime
    https://cilium.io
  - PcapRecorder: XDP based BPF clone of Tcpdump, filters generated at runtime
    https://cilium.io/blog/2021/05/20/cilium-110#pcap
- Locks down user space runtime patching
- Locks down user space code generation based on configuration

Goal

Block malicious actors Eve from loading and manipulating programs while allowing code generation and optimizations.

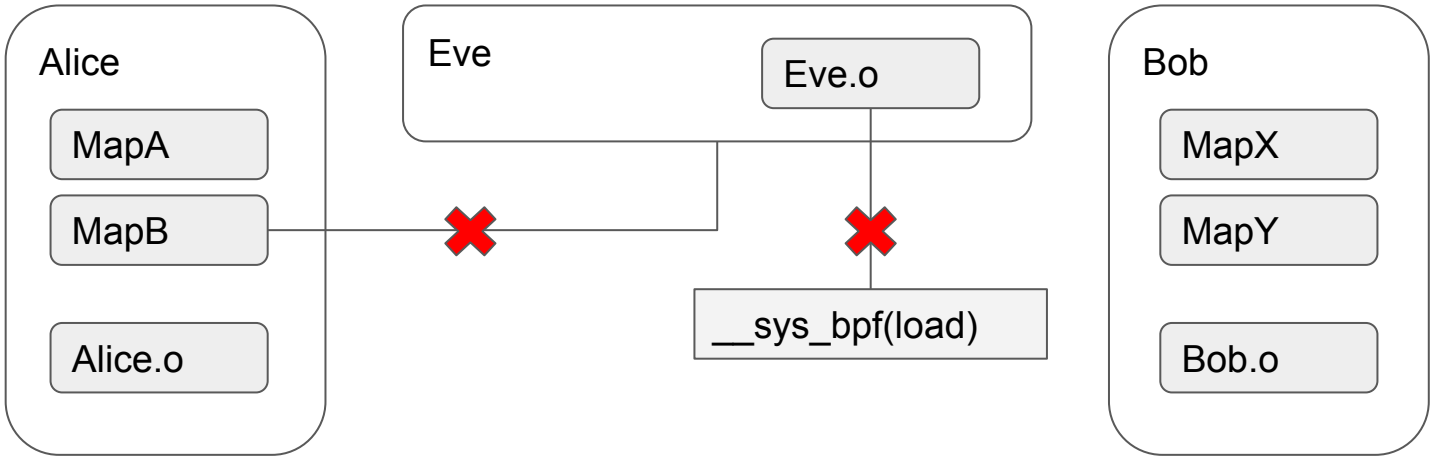
Block imposter Alice from loading and manipulating programs while allowing code generation and optimizations.



Goal: Example 1

- Given two BPF Users: Program Alice and Program Bob
- Given a malicious program Eve

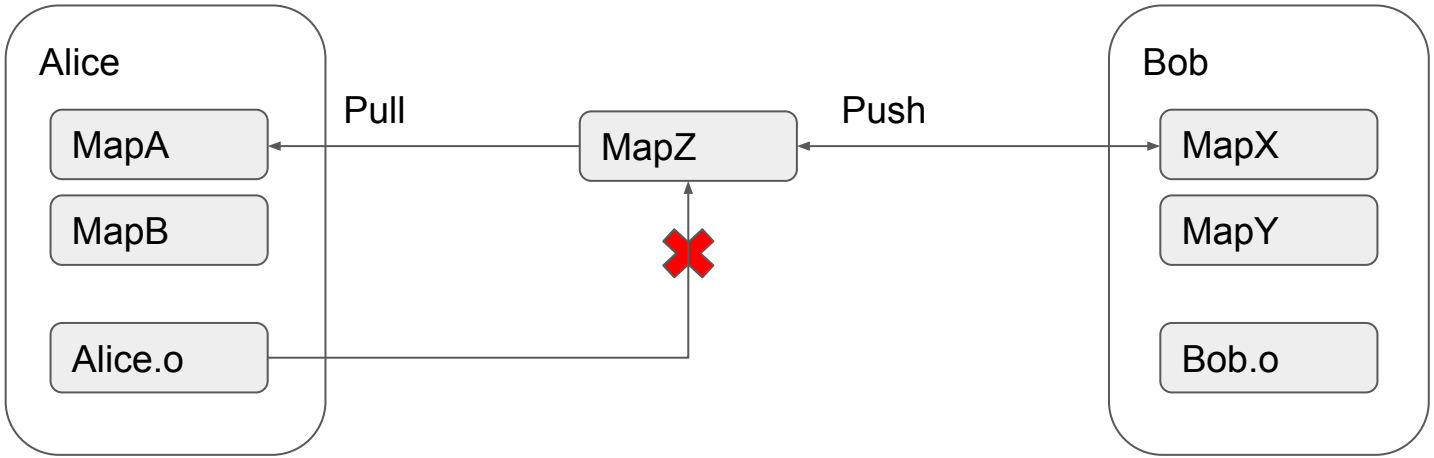
Goal: Ensure Eve can not load programs or read/write maps



Goal: Example 2

- Given two BPF Users: Program Alice and Program Bob
- Assume MapM is shared between Alice and Bob
  - Common for systems with many BPF applications or cross system tasks (application aware firewall)
  - Allows exposing Stable versioned Map APIs to system tools
  - Unix philosophy of BPF programs -- avoiding the BPF monolith

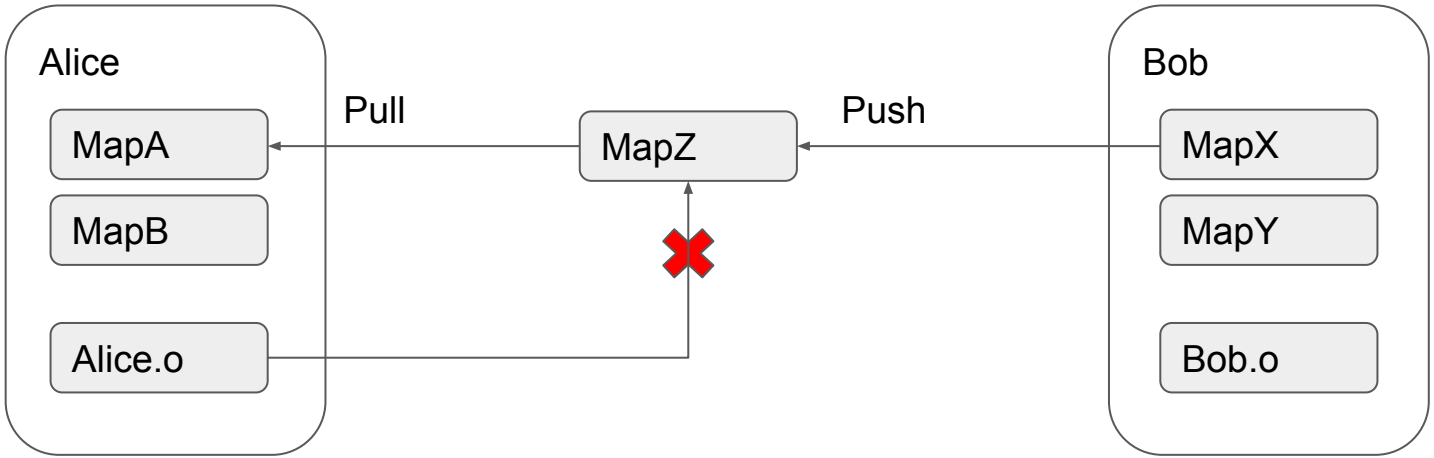
Goal: Ensure bugs in Alice can not impact Bob (RO/RW map policy)



Goal: Example 3

- Given two BPF Users: Program Alice and Program Bob
- Assume Alice is buggy or runtime compromise
  - Alice may attempt to load incorrect programs
  - How can we minimize or mitigate the impact under a dynamic code generation model.

Goal: Secure dynamic code generation



Tools

- Fsverity: read-only file-based authenticity protection
  . “By itself, the base fs-verity feature only provides integrity protection, i.e. detection of accidental (non-malicious) corruption.”
  https://www.kernel.org/doc/html/latest/filesystems/fsverity.html.
  https://lwn.net/Articles/763729/

- IMA: Integrity Measurement Architecture (alternatively)

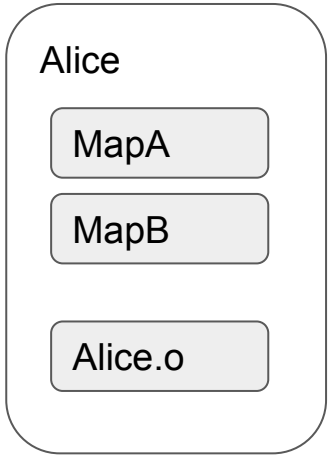
- Libbpf and cilium/eBPF

- BPF *the linux superpower*

Tools: Fsverity

- Fsvirty: read-only file-based authenticity protection
- Kernel support
  - CONFIG_*VERITY*
- User space component ‘fsverity’
  - Enable FS: `mk2efs -O verity …`
  - fsverity enable FILE
- **Any reads of corrupted data will fail**
- Key ring support:
  CONFIG_FS_VERITY_BUILTIN_SIGNATURE=y

$ fsverity enable Alice



Corrupted alice will not launch

Tools: BPF read Fsverity Hash

Ioctl:

```
int fsverity_ioctl_measure(struct file *filp, void __user *_uarg) {
  const struct fsverity_info *vi = fsverity_get_info(file);
  copy_to_user(uarg->digest, vi->file_digest, digest_size);
}
```

BPF Directly: Audit this works well enough

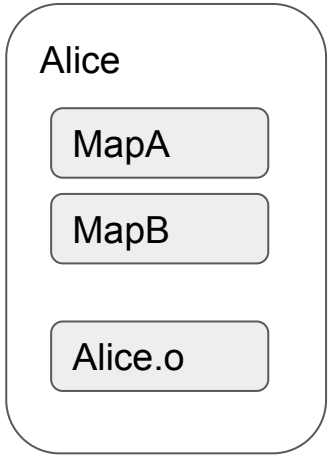
```
struct vsverity_info {
  struct merkle_tree_params tree_params;
  u8 root_hash[FS_VERITY_MAX_DIGEST_SIZE];
  u8 file_digest[FS_VERITY_MAX_DIGEST_SIZE];
  const struct inode *inode;
}

struct vsverity_info *bpf_get_fsverity_info(struct file *file) {
    struct inode *i = _(&file->inode);
    return smp_load_acquire(_(&i->i_verify_info)); // mb()
}
```

BPF Helper: bpf_fsverity_verify_hash(): (Open question)

- Is it useful to verify hash if open would fail otherwise?

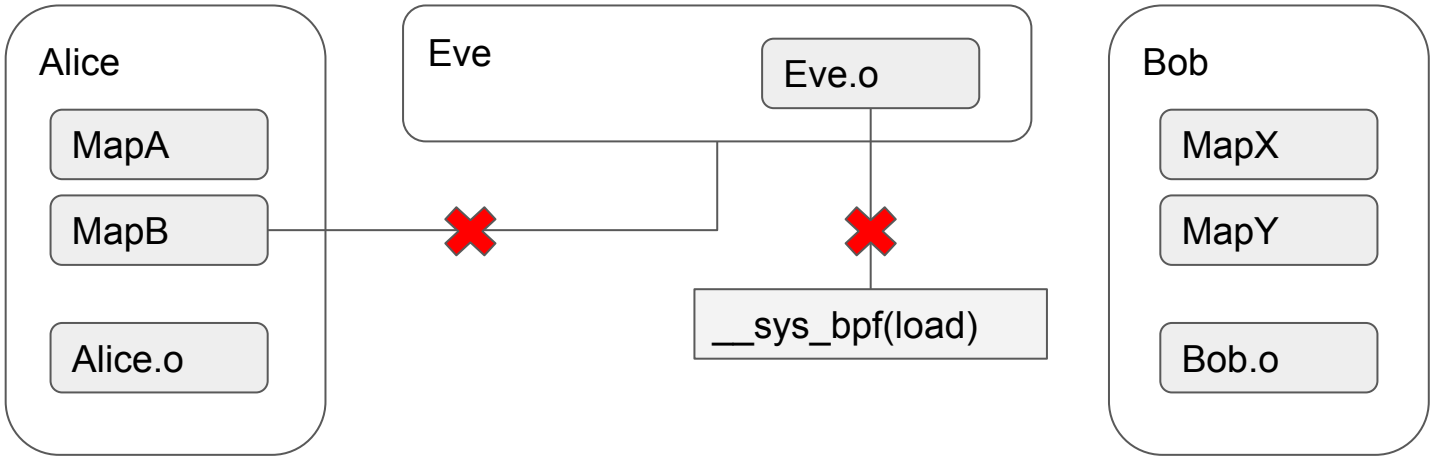
$ fsverity enable Alice



Corrupted alice will not launch

Solution 1: BPF/File verification Approach

- Sign Alice and Bob executables.
- Sign Alice and Bob collateral (templates, object files, generating tools, etc.)
- At exec verify Alice, Bob and at open verify collateral
- Mark Alice and Bob as “authorized” __sys_bpf(load) users in task_struct
- “Verify” task_struct authorized attribute at __sys_bpf(load)



Solution 1: BPF/File verification Approach

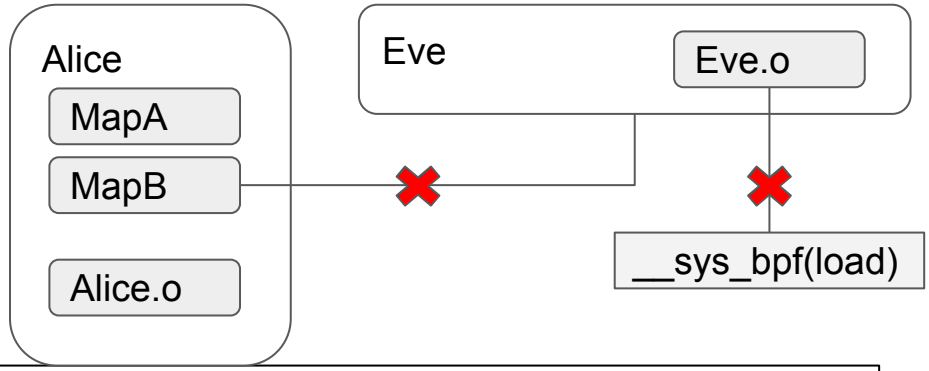
- Sign Alice and Bob executables.
- Sign Alice and Bob collateral (templates, generating tools, etc.)
- At exec verify Alice, Bob and at open verify collateral
- Mark Alice and Bob as “authorized” __sys_bpf(load) users in task_struct
- “Verify” task_struct “authorized” attribute at __sys_bpf(load)

```
# watchbpf --enforce alice.yaml
 . process: eve   pid: 284 pod: eve op: exec  action: BPFDenied
 . process: eve   pid: 284 pod: eve op: load   type: xdp name: from-netdev action: BPFDenied
 …
 . process: alice pid: 262 pod: alice op: exec action: ProgApproved,MapsDenied
 . process: alice pid: 262 pod: alice op: load  type: xdp name: from-netdev action: Approved
 …
 . process: eve  pid: 262 pod: alice op: update name: ct_map_tcp4 action: Denied
 . process: eve  pid: 262 pod: alice op: update name: ct_map_tcp4 action: Denied
```

Solution 1: BPF/File verification Approach

Example 1:

Ensure Eve can not load programs or read/write maps

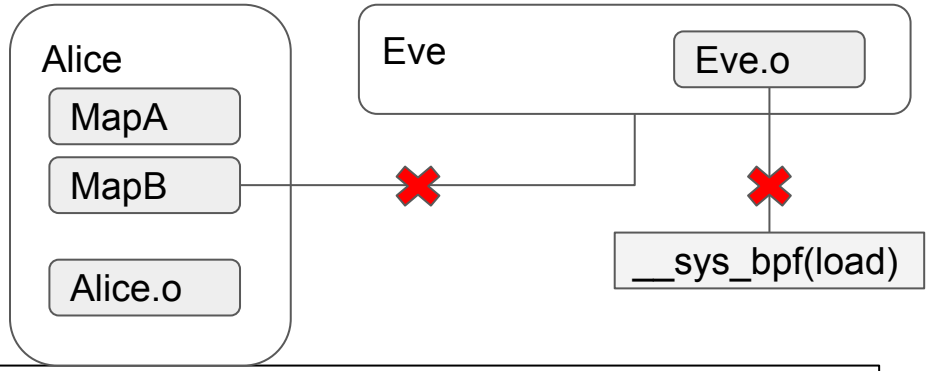



```
# watchbpf --enforce alice.yaml
 . process: eve   pid: 284 pod: eve op: exec  action: BPFDenied
 . process: eve   pid: 284 pod: eve op: load   type: xdp name: from-netdev action: BPFDenied
 …
 . process: alice pid: 262 pod: alice op: exec action: ProgApproved,MapsDenied
 . process: alice pid: 262 pod: alice op: load  type: xdp name: from-netdev action: Approved
 …
 . process: eve  pid: 262 pod: alice op: update name: ct_map_tcp4 action: Denied
 . process: eve  pid: 262 pod: alice op: update name: ct_map_tcp4 action: Denied
```

Solution 2: BPF/File verification Approach

Example 2:

Ensure bugs in Alice can not impact Bob (RO/RW map policy)

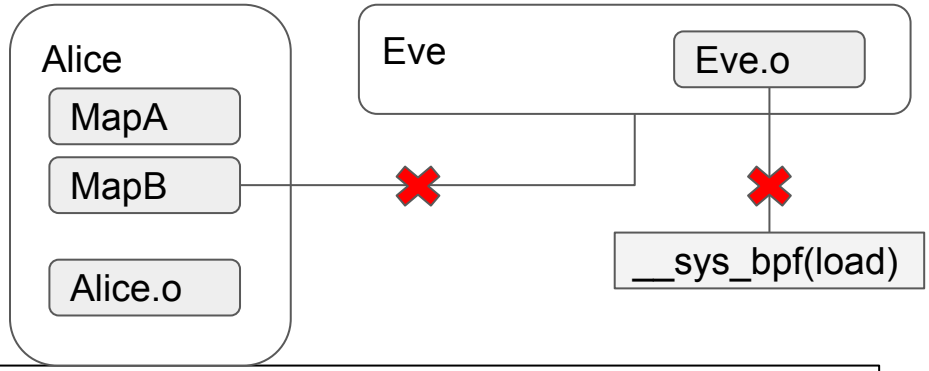



```
# watchbpf --enforce alice.yaml
 . process: eve   pid: 284 pod: eve op: exec  action: BPFDenied
 . process: eve   pid: 284 pod: eve op: load   type: xdp name: from-netdev action: BPFDenied
 …
 . process: alice pid: 262 pod: alice op: exec action: ProgApproved,MapsDenied
 . process: alice pid: 262 pod: alice op: load  type: xdp name: from-netdev action: Approved
 …
 . process: eve  pid: 262 pod: alice op: update name: ct_map_tcp4 action: Denied
 . process: eve  pid: 262 pod: alice op: update name: ct_map_tcp4 action: Denied
```

BPF/File verification Approach

Example 3: Alice with runtime corruption

So what can we do?

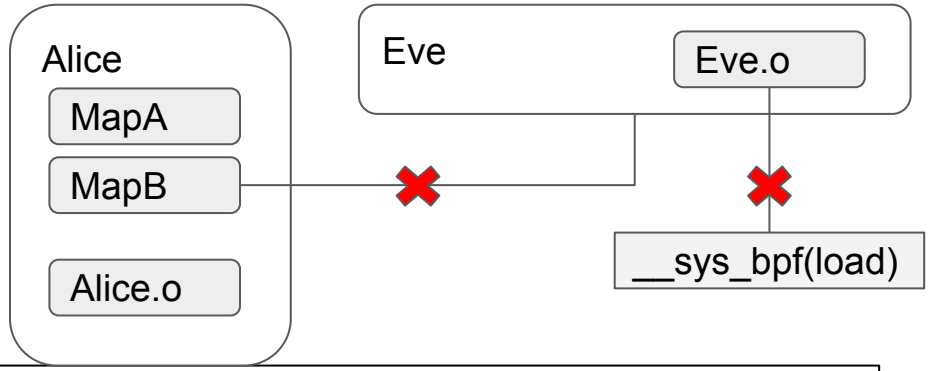



```
# watchbpf --enforce alice.yaml
 …
 . process: alice pid: 262 pod: alice op: exec action: ProgApproved,MapsDenied
< alice corrupted >
 . process: alice pid: 262 pod: alice op: load  type: xdp name: from-netdev action: Approved
 …
```

BPF/File verification Proposal

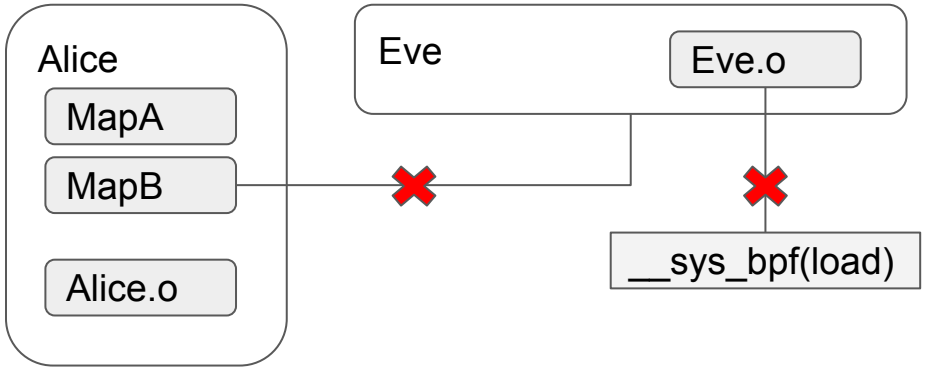
Example 3: Alice with runtime corruption

: Thought experiment, corrupt program loads signed object file.




```
# watchbpf --enforce alice.yaml
…
. process: alice pid: 262 pod: alice op: exec action: ProgApproved,MapsDenied
< alice is corrupt, alice.o signed >
. process: alice pid: 262 pod: alice op: load  type: xdp name: from-netdev action: Approved
…
```

BPF/File verification Proposal



Example 3: Alice with runtime corruption

Thought experiment, corrupt program loads signed object file.

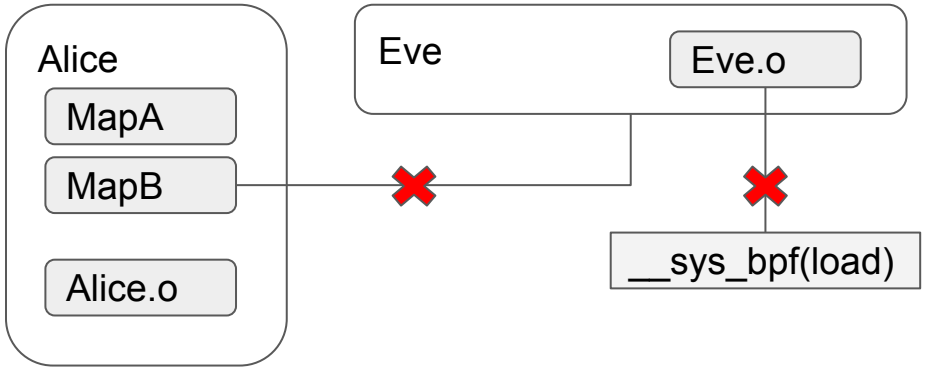
: Network application
- Delete Firewall, redirect traffic, eavesdrop, etc.

: Observability
- Manipulate maps and tail calls, /dev/null events.

: Security application
- Delete/Add policy, remove checks, etc.
- Use incorrect attach points

Opinion: Perhaps not as problematic as a incorrect BPF program, but critical failure none the less.

BPF/File verification Proposal



Example 3: Alice with runtime corruption

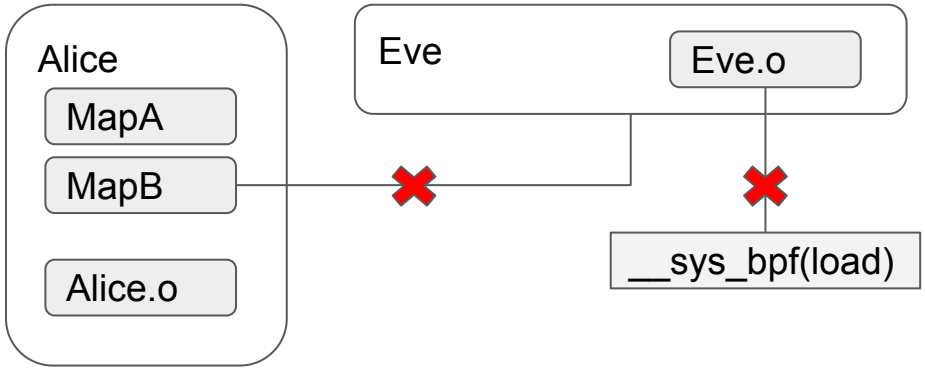
Proposal, given enough information about the program being loaded we can create a policy to allow or deny it.

Info: calls, kernel memory reads, map read, map writes, etc.

How:

.Allow unsigned programs without write_user() calls
.Allow unsigned programs without kernel memory reads
.Allow program that only writes unpinned maps
...

BPF/File verification Proposal



Example 3: Alice with runtime corruption

Proposal, given enough information about the program being loaded we can create a policy to allow or deny it.

How:

1. Verifier collects info while verifying program
2. Calls BPF program with extra program metadata
3. BPF program allows or disallows program based on metadata

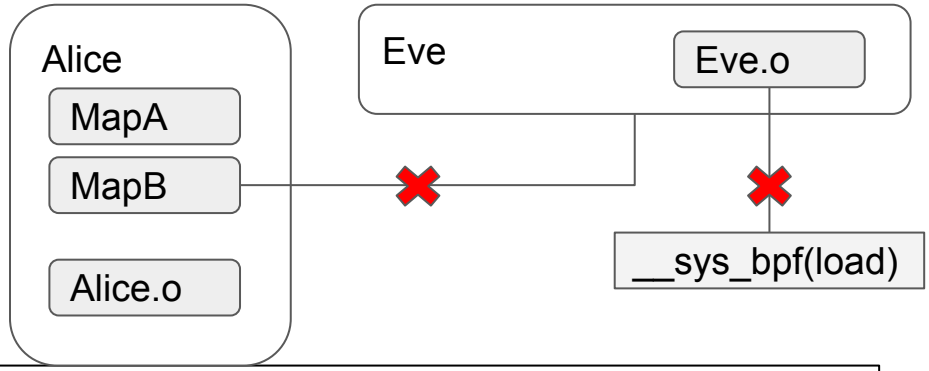
Can be combined with signature checking or not.

Will BPF attributes help? Could pass conditions down for verification.

BPF/File verification Approach

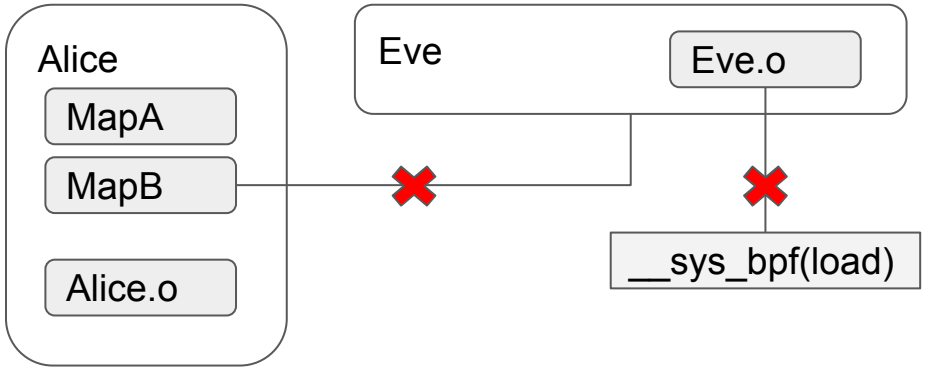
Open Questions:

“Verify” - BPF helper to provide key ring attributes?
“Authorized” - Secure read-only map of policy?




```
# watchbpf --enforce alice.yaml
 . process: eve   pid: 284 pod: eve op: exec  action: BPFDenied
 . process: eve   pid: 284 pod: eve op: load   type: xdp name: from-netdev action: BPFDenied
 …
 . process: alice pid: 262 pod: alice op: exec action: ProgApproved,MapsDenied
 . process: alice pid: 262 pod: alice op: load  type: xdp name: from-netdev action: Approved
 …
 . process: eve  pid: 262 pod: alice op: update name: ct_map_tcp4 action: Denied
 . process: eve  pid: 262 pod: alice op: update name: ct_map_tcp4 action: Denied
```

BPF/File verification Approach



Next Steps:

Control Plane, how to manage Verification?

Do we need access to a key ring?

Do we need more helpers? Trigger measure?

IMA hooks exist can we use them.

Summary:

0. BPF program signing appears to be incompatible with much of the useful BPF tooling

1. Application signing (e.g. signing the tools instead of signing the BPF program) covers many cases.

2. Corrupt applications can break systems from userspace only

3. Improved visibility into BPF program launch info may allow runtime security policy in many cases providing similar levels of security.

Thank You!

Questions?

John Fastabend <john.fastabend@gmail.com>
Isovalent, Cilium.io