Towards Functional Verification of eBPF Programs

Dana Lu, Boxuan Tang,
Michael Paper and Marios Kogias

IMPERIAL

eBPF '24 Workshop
August 4, 2024

SIGCOMM 2024
— SYDNEY —

eBPF Deployed in Important Places

cilium

Falco

eBPF Programs May Be Written By Others

eBPF Marketplaces

AI Code Generation

There is a need to verify the behaviour of eBPF programs

eBPF Verifier

Safety

- No unsafe memory access
- Termination

Functionality

- Return value
- Side effects

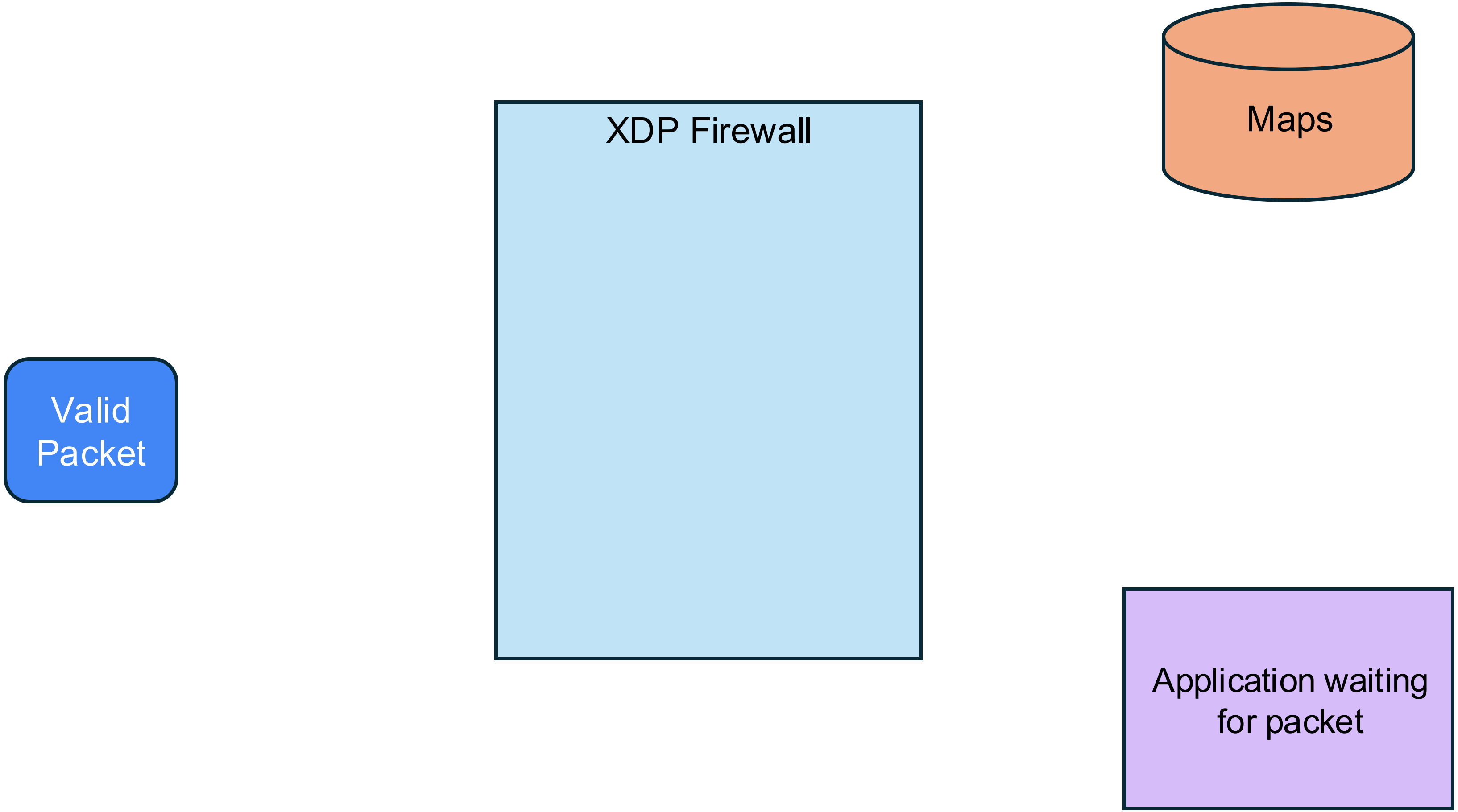
Verification of Individual Programs is Insufficient

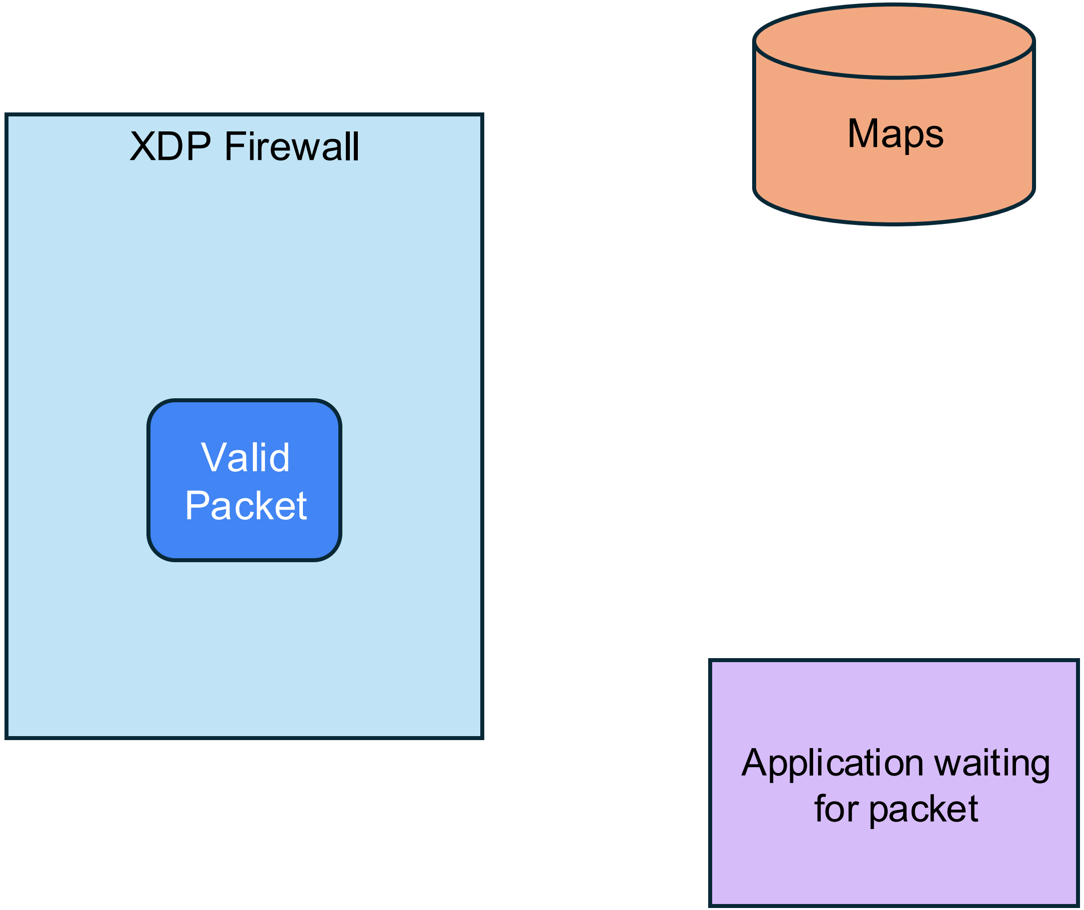
Behaviour of eBPF Programs is highly dependent on external interactions

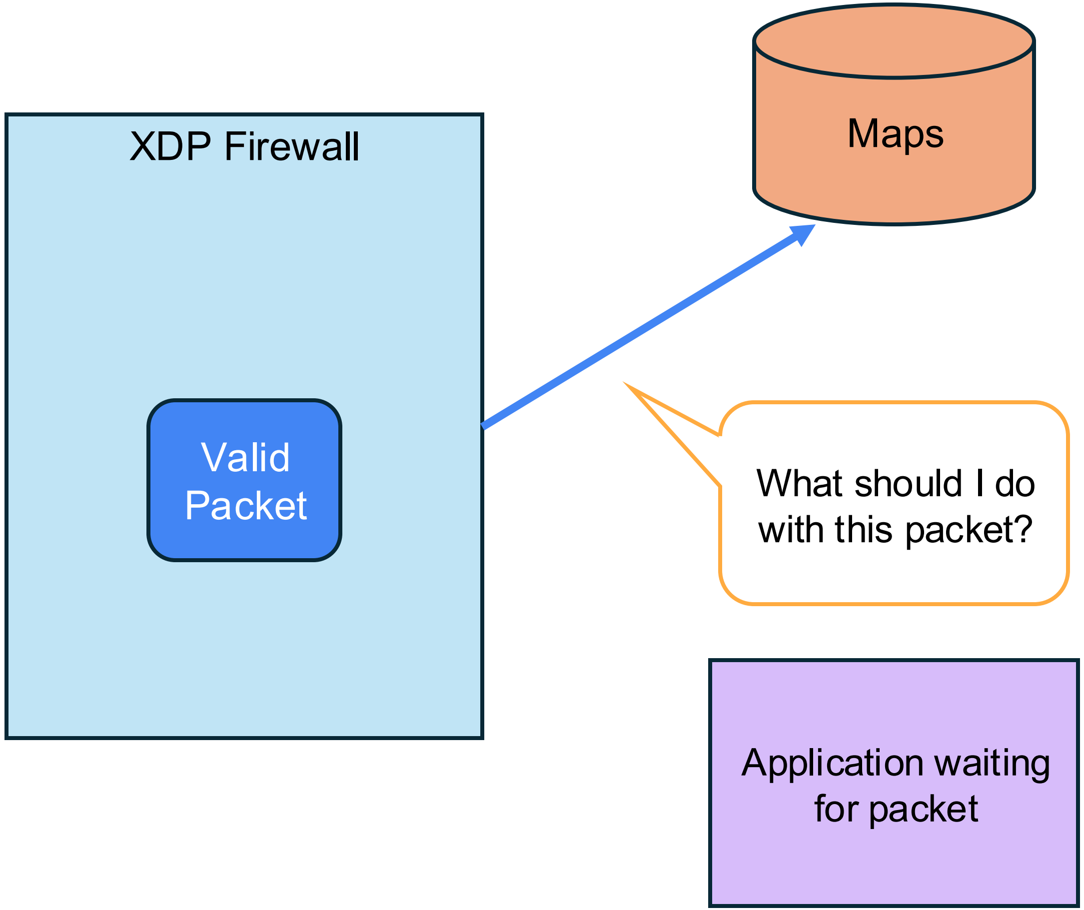
Verification of Individual Programs is Insufficient

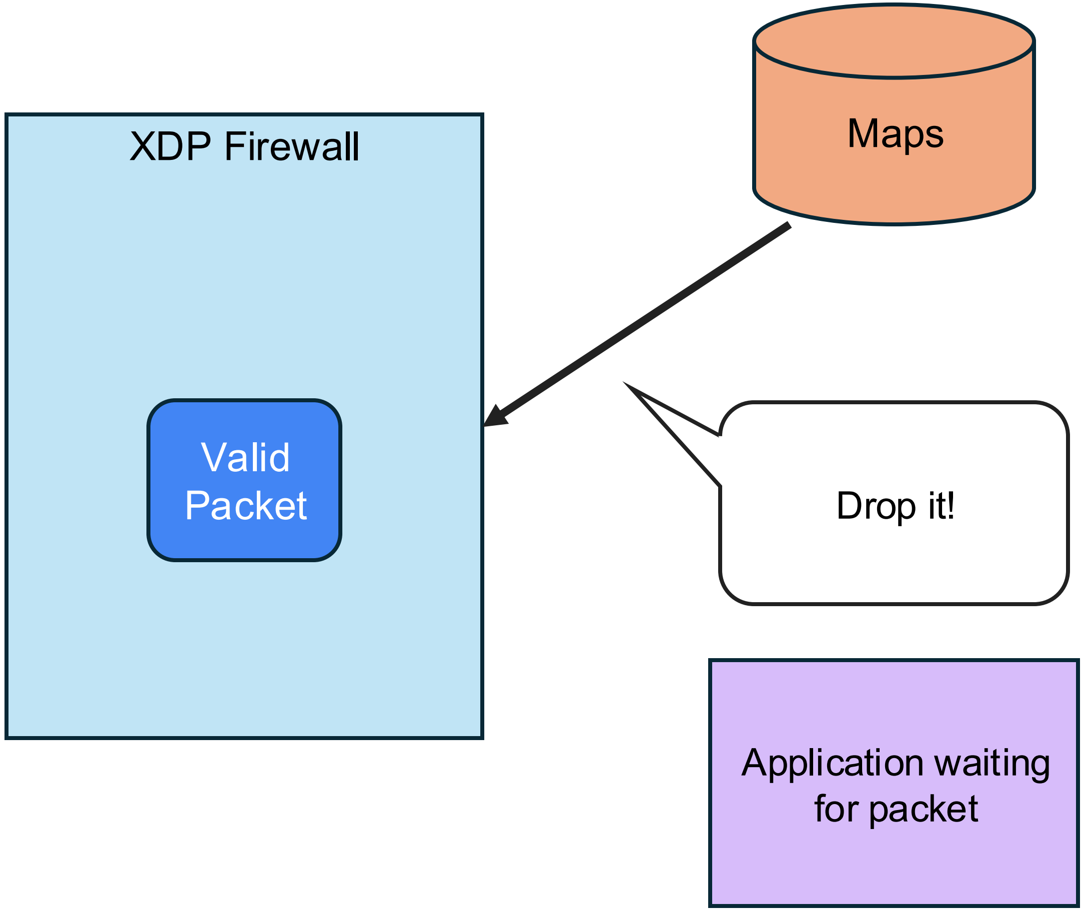
Behaviour of eBPF Programs is highly dependent on external interactions

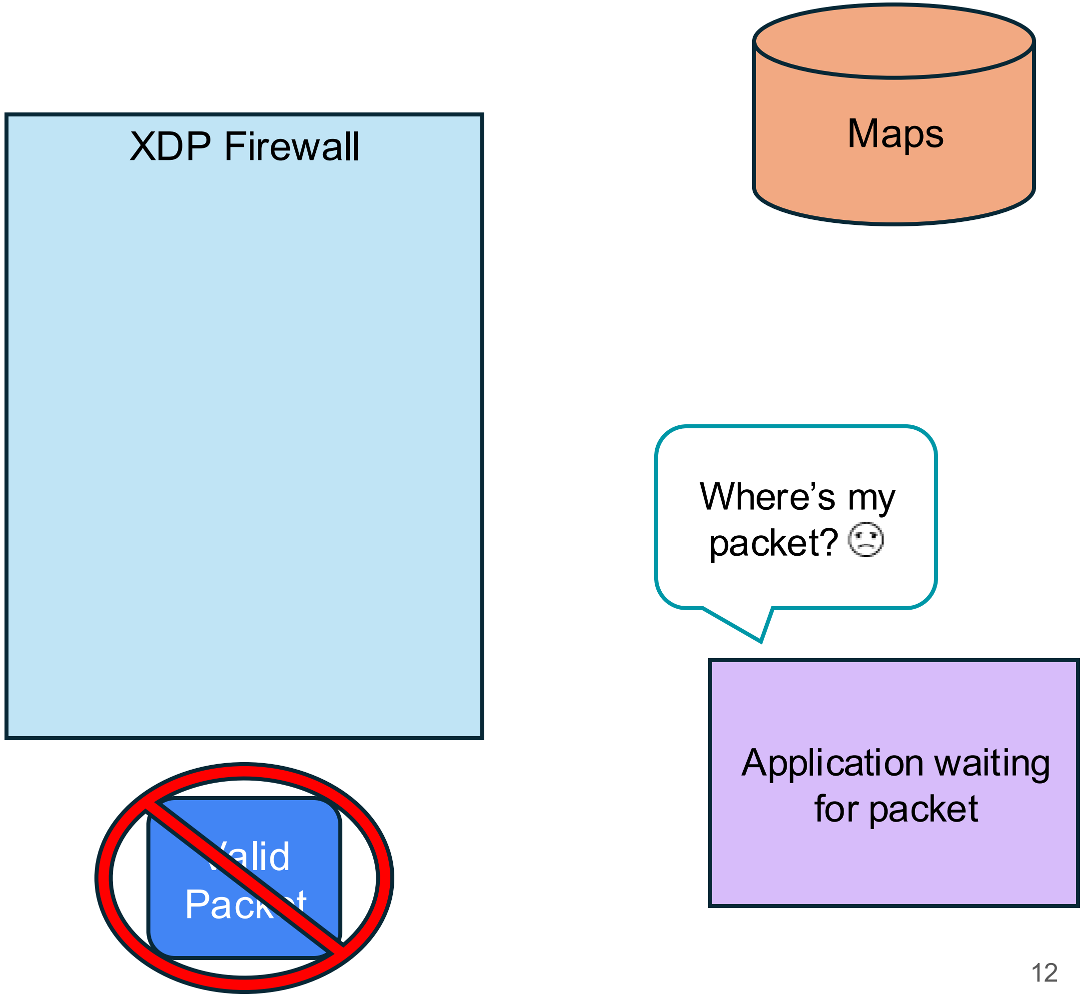
BPF Maps
Contents


Maps
XDP Firewall
Valid
Packet
Application waiting
for packet


XDP Firewall
Valid Packet
Maps
Application waiting for packet

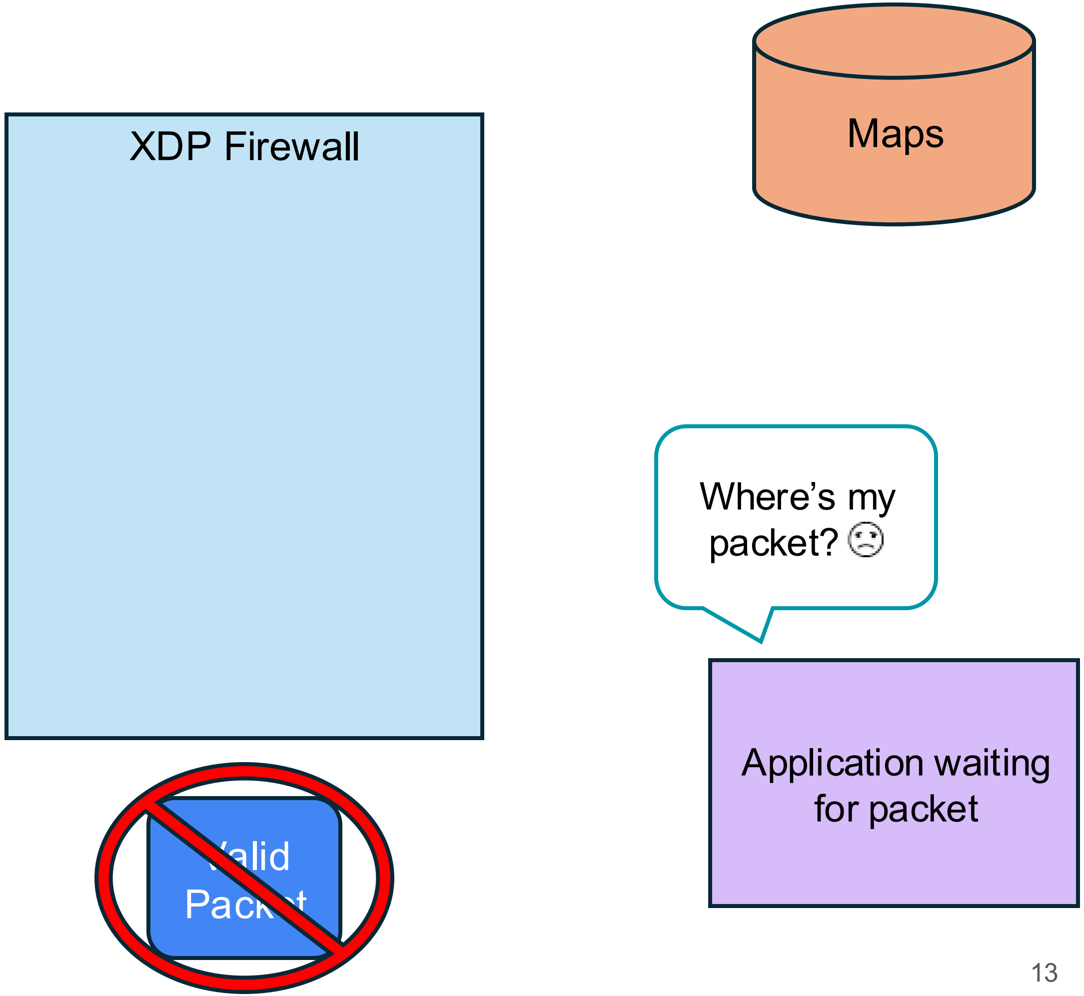

XDP Firewall
Valid Packet
Maps
What should I do with this packet?
Application waiting for packet


XDP Firewall
Valid Packet
Maps
Drop it!
Application waiting for packet



XDP Firewall
Maps
Where's my packet? ☹
Application waiting for packet
Valid Packet

Problem: maps influence the behaviour of programs

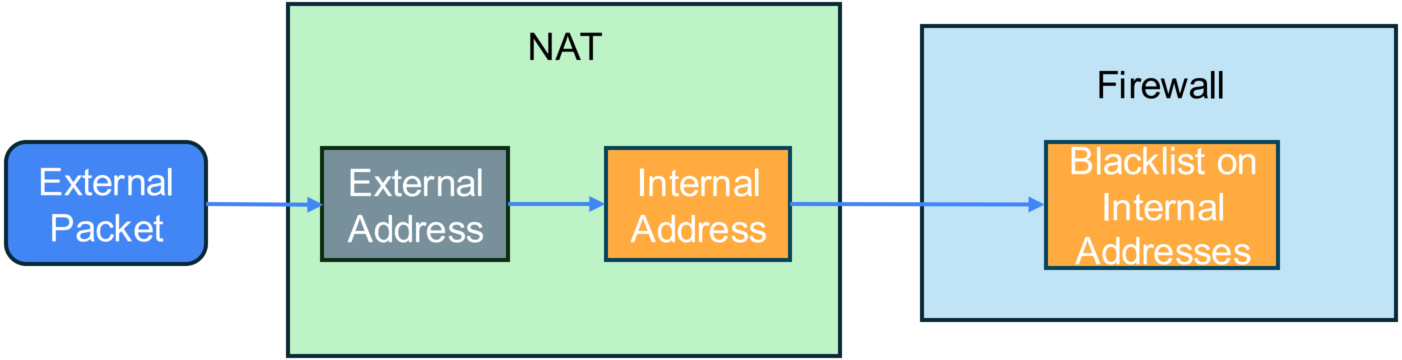


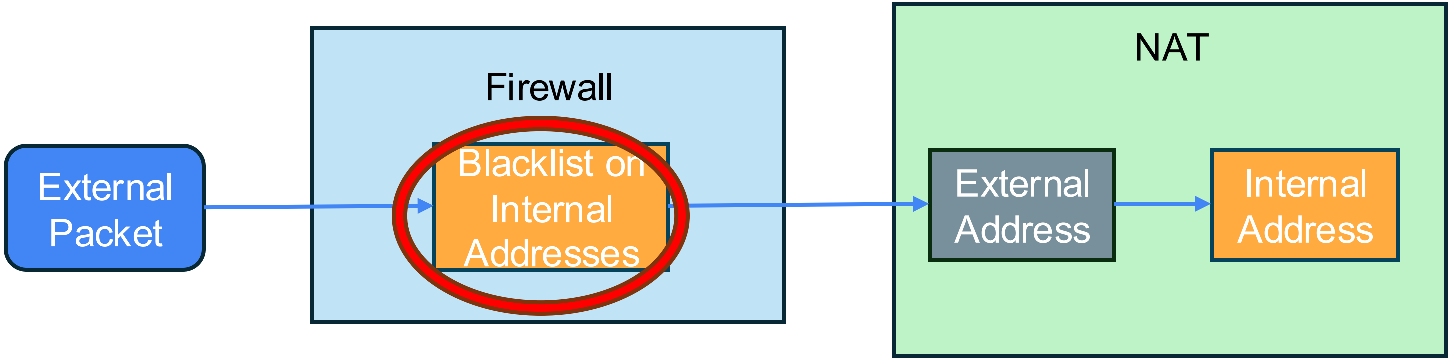
Verification of Individual Programs is Insufficient

Behaviour of eBPF Programs is highly dependent external interactions

BPF Maps Contents

Other eBPF Programs


External Packet
NAT
External Address
Internal Address
Firewall
Blacklist on Internal Addresses



Problem: the ordering of programs can change the behaviour

Also important to analyse a program's external interactions

Introducing DRACO

- Verify behaviour of individual eBPF programs against a specification format

- Analyse how eBPF program's external interactions affects program behaviour

*Initially focus on XDP programs

DRACO Insight

- Loaded eBPF programs have to pass the kernel verifier
  - Bounded number of execution paths
  - Ideal for Exhaustive Symbolic Execution (ESE)
  - Avoids path explosion

Symbolic Execution

- Symbolic Execution is a technique to explore possible execution paths in a program
    - Path explosion from branching and loops

- KLEE is a Symbolic Execution Engine on the LLVM level

DRACO: Verifying Individual Programs

- External Specifications

- Integrated Specifications

Integrated Specifications

- Embedded throughout the eBPF program
- Temporal assertions across two points of the program
  - Where the assertion is located
  - When the program terminates
- ESE in KLEE explores and makes relevant assertions in all paths

Integrated Specification Examples

```
ethernet = data ;
BPF_ASSERT_CONSTANT(ethernet, sizeof(*ethernet));
```

BPF_ASSERT_CONSTANT asserts a memory location remains constant

```c
    BPF_ASSERT_IF_ACTION_THEN_NEQ(XDP_DROP, &(ip->protocol), __u8, IPPROTO_TCP);

    nh_off +=sizeof(*ip);
    if (data + nh_off  > data_end)
        goto EOP;

    if(ip->protocol != IPPROTO_TCP){
            goto EOP;
    }

    BPF_ASSERT_RETURN(XDP_TX);

    int key = ip->saddr;
    int value = 1;
    bpf_map_update_elem(&example_map, &key, &value, 0);

    BPF_RETURN(XDP_TX);

EOP:
    BPF_RETURN(XDP_DROP);
```

BPF_ASSERT_IF_ACTION_THEN_EQ asserts that if an XDP action is returned the given memory location must not be equal to the given value

BPF_ASSERT_RETURN asserts that the given XDP action must be returned

Read paper for more examples

Driver Program for Verifying Integrated Specifications

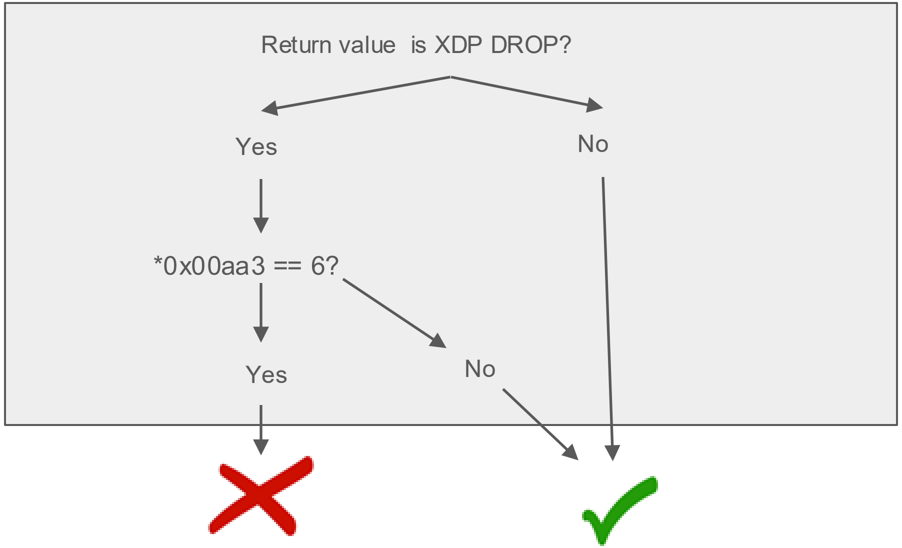
1. System of arrays is declared by driver program

2. Relevant Information is enqueued onto arrays when execution passes through assertion:
memory locations, sizes, bytes, return values

Driver Program for Verifying Integrated Specifications

3. When program terminates, go through enqueued assertions

```
BPF_ASSERT_IF_ACTION_THEN_NEQ(XDP_DROP, &(ip->protocol), __u8, IPPROTO_TCP);
```



External Specification

- Executable program written in any language compliable to LLVM
- Implement the same functionality as the eBPF program
  - But not the same safety and performance requirements
- ESE in KLEE to verify program against specification
- Specification can be either *full* or *partial*
  - Return value
  - Changes to BPF Maps
  - Changes to network packet

~~eBPF Verifier~~

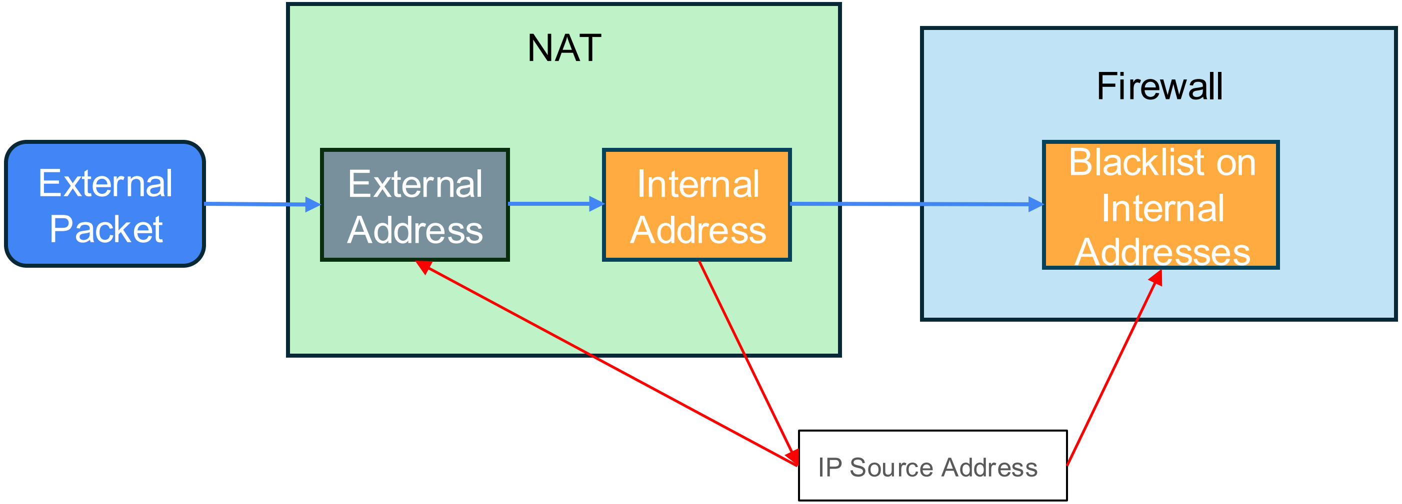
Read paper
for details

DRACO: Analysing Program Interactions

- Determine if the order of execution of two eBPF programs matters
- Identify dependencies between BPF maps
- Identify how BPF map contents affects program branching

Checking if Order of Execution Matters

- BPF Map and Packet Data
- Written to by both programs or
  Written to by one program and read by the other



Checking if Order of Execution Matters

```
define dso_local i32 @ xdp_prog(%struct xdp_md* %ctx) #5 section "xdp_
XDP " !
   ...
   %3 =                                                  ign 8
   %data                                                 dp_md* %3,
i32 0,
   %4 =
   %conv3 = zext i32 %4 to i64
   %5 = inttoptr i64 %conv3 to i8
   store i8* %5, i8** %data, align 8
}
```

Extend KLEE to:
For every path,

1. Check if memory location is BPF map / network packet
2. Add to ReadSet / WriteSet accordingly
3. Check if there is any overlap in both programs’ sets

DRACO: Analysing Program Interactions

- Determine if the order of execution of two eBPF programs matters
- Identify dependencies between BPF maps
- Identify how BPF map contents affects program branching

Read paper
for details

Evaluation Criteria

- Specification LOC 
- Time 

Evaluation of DRACO Verification

Program	LOC	Type	Spec	Paths	Time
hXDP FW	686	Full	27	64	6.93s
hXDP FW	686	Full	18	4	3.45s
Fluvia	156	Partial	4	23	23.35s
Katran	4244	Partial	17	10	71.24s
CRAB	365	Assert	20	5	1.90s

Read paper for details

Concluding Remarks

- Verified eBPF programs are suitable for further verification and analysis using symbolic execution
- Verification can be integrated as part of deployment pipeline
- eBPF programs’ behaviour are dependent on external interactions, it is beneficial to analyse the interactions of eBPF programs during development

Thank you!