

LINUX PLUMBERS 2026 • EBPf

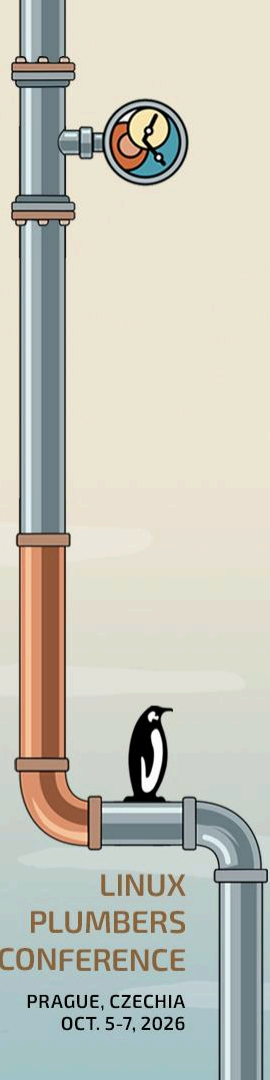
Evolution of Value Tracking in the BPF Verifier

Shung-Hsi Yu SUSE



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

A vertical pipe with a gauge and a penguin. The pipe is grey with orange sections. A gauge is attached to the top, and a penguin is perched on a horizontal section near the bottom.

SECTION 1

The Beginning

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

```
tcpdump 'tcp dst port 80'
```

a packet filter. the kernel runs it on every packet.

```
// tcpdump 'tcp dst port 80'

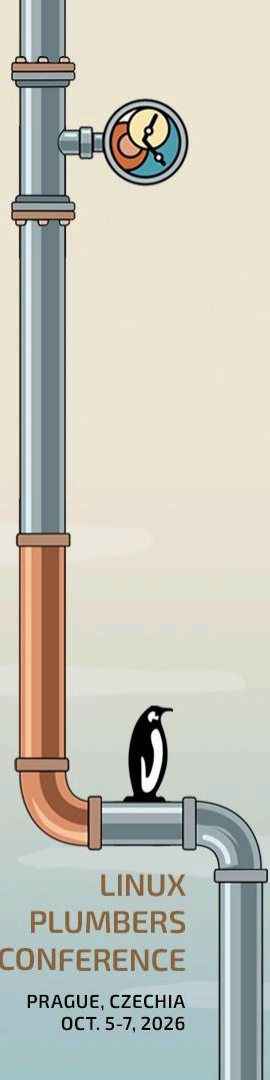
r0 = *(u8 *)skb[14]           // BPF_ABS: IP version, header length
r0 &= 0xf
r0 <= 2                       // header length: 20 to 60 bytes
r7 = r0
r0 = *(u16 *)skb[r7 + 16]     // BPF_IND: TCP dst port
```

skb[...] reads the packet through r6, implicitly. r7 is just an offset.

```
off = IMM + SRC;                // 16 + r7
ptr = bpf_load_pointer(skb, off, 2, &tmp);
if (likely(ptr != NULL)) {
    BPF_R0 = get_unaligned_be16(ptr);
    CONT;
}
return 0;
```

every load checked as it runs. out of range and the program just stops.

**Checked once,
at load**



V3.18

2014

17a5267067f3

LINUX
PLUMBERS
CONFERENCE

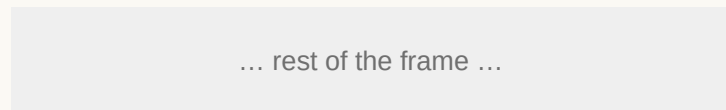
PRAGUE, CZECHIA
OCT. 5-7, 2026

```
char buf[8];  
len = 8;    // CONST_IMM 8  
bpf_probe_read(buf, len, src);
```

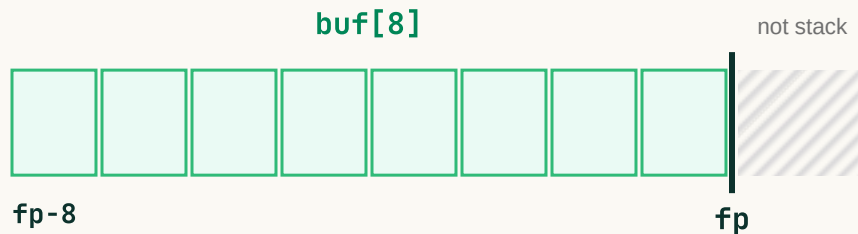
8 bytes of stack. and a constant length.

```
r1 = r10
r1 += -8           // buf: fp-8
r2 = 8           // len: CONST_IMM 8
r3 = *(u64 *)(r10 - 24) // src
call bpf_probe_read
```

stack frame · MAX_BPF_STACK = 512



fp-512



len = 8



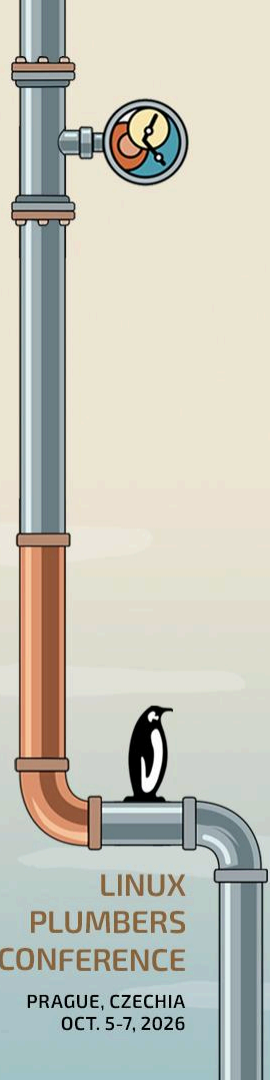
✓ fp-8 + 8 = fp: fits

8 bytes: the access stops exactly at fp.

MAX_BPF_STACK = 512

```
r0 = map value          // PTR_TO_MAP_VALUE  
r2 = *(u64 *)(r0 + 0)   // UNKNOWN_VALUE
```

but read the length from a map value, and it is unknown.



SECTION 2

Initial Tracking

LINUX
PLUMBERS
CONFERENCE

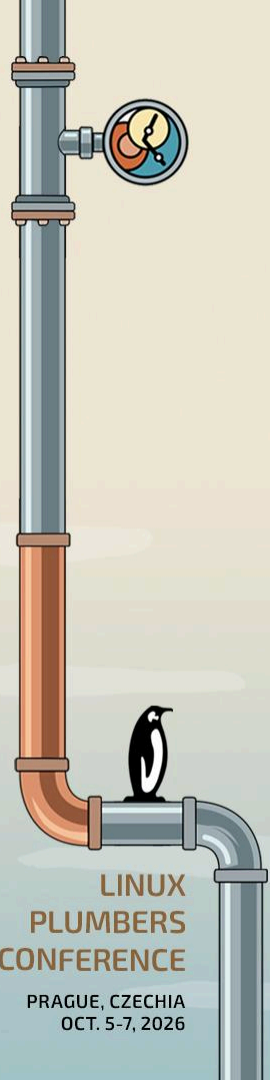
PRAGUE, CZECHIA
OCT. 5-7, 2026

```
char buf[8];
u64 len = val->len;    // read from a map value

if (len > 8)
    return 0;
if (len < 1)
    return 0;

bpf_probe_read(buf, len, src);    // always 1 .. 8
```

but it is always 1 .. 8. a constant can't say that.



V 4 . 9

2016

484611357c19 · Josef Bacik

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Track a range

```
s64 min_value;  
u64 max_value;
```

one signed, one unsigned. remember that.

PROGRAM

```
r2 = *(u64 *) (r10 - 16)
```

load from the stack: anything

POSSIBLE VALUES OF R2

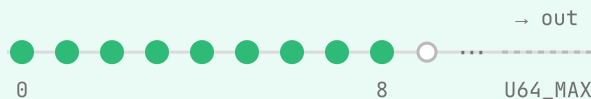


VERIFIER STORES

unknown

```
if r2 > 8 goto out
```

fall-through: $r2 \leq 8$



max_value = 8

BPF_JGT. taken: $r2 > 8$. fall-through: $r2 \leq 8$, so the max is 8.

```
r0 = map value           // 1 byte, say
r1 = *(u64 *)(r10 - 16)  // unknown
r2 = -1                  // U64_MAX
if r1 > r2 goto out
if r1 s> 1 goto out
r0 += r1
*(u8 *)(r0 + 0) = 0
out:
exit
```

one unsigned check, one signed check. then a store.

PROGRAM

```
r1 = *(u64 *) (r10 - 16)
```

load from the stack: anything

```
if r1 > r2 goto out
```

r2 = U64_MAX. unsigned, never taken

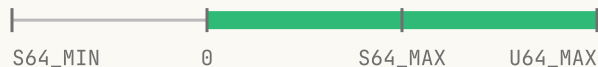
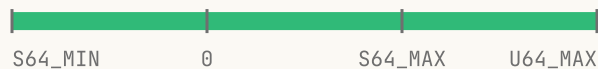
```
if r1 s> 1 goto out
```

signed. fall-through: r1 ≤ 1

```
r0 += r1; *(u8 *) (r0 + 0) = 0
```

r1 is 0 .. 1: in bounds

R1, AS THE VERIFIER SEES IT



VERIFIER STORES

unknown

min_value = 0

min_value = 0

max_value = 1

min_value = 0

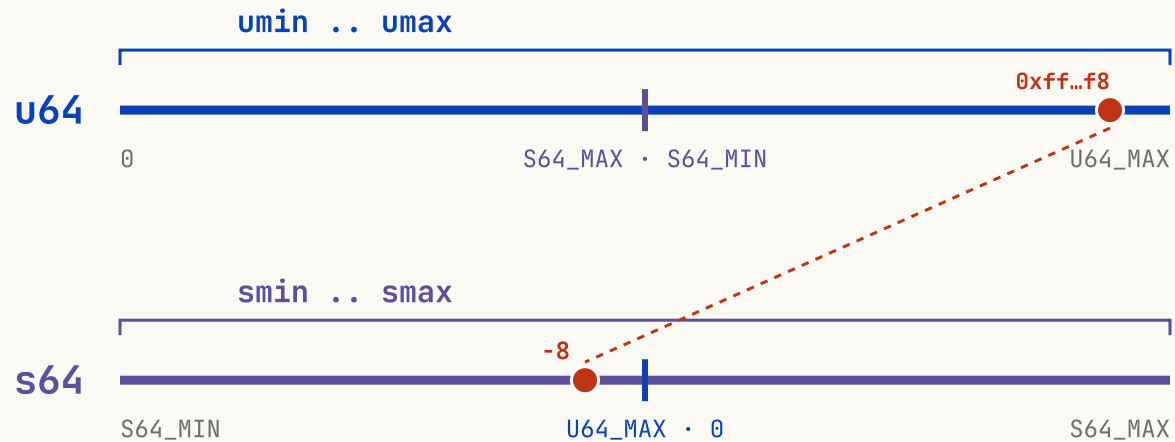
max_value = 1



so r1 is 0 .. 1, and the store is in bounds.

-8

what it actually holds



two lines. same bits. both kept.

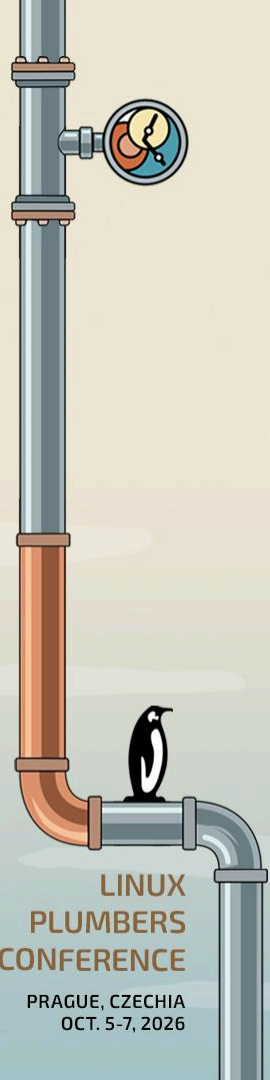


SECTION 3

Rework

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



V 4 . 1 4

2017

b03c9f9fdc37 • Edward Cree

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Track both

PROGRAM

```
r1 = *(u64 *) (r10 - 16)
```

load from the stack: anything

```
if r1 > r2 goto out
```

r2 = U64_MAX. unsigned, never taken

```
if r1 s> 1 goto out
```

signed. fall-through: r1 ≤ 1

```
r0 += r1; *(u8 *) (r0 + 0) = 0
```

umax = U64_MAX: out of bounds

R1, AS THE VERIFIER SEES IT



VERIFIER STORES

unknown

umax = U64_MAX

smax = S64_MAX

umax = U64_MAX

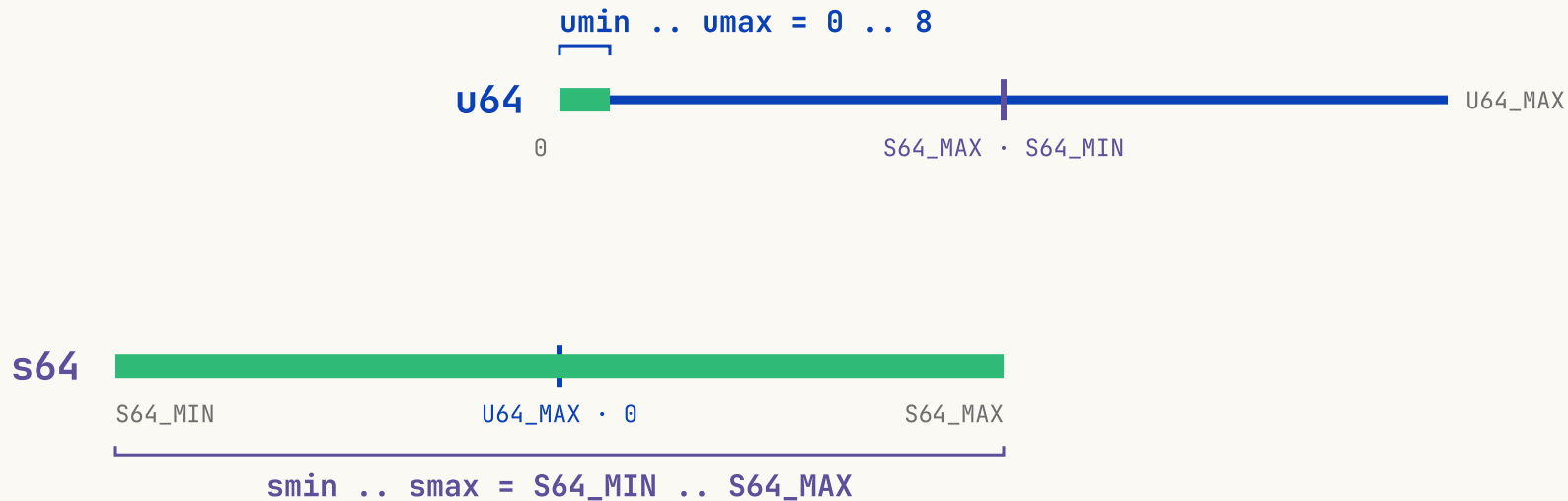
smax = 1

umax = U64_MAX

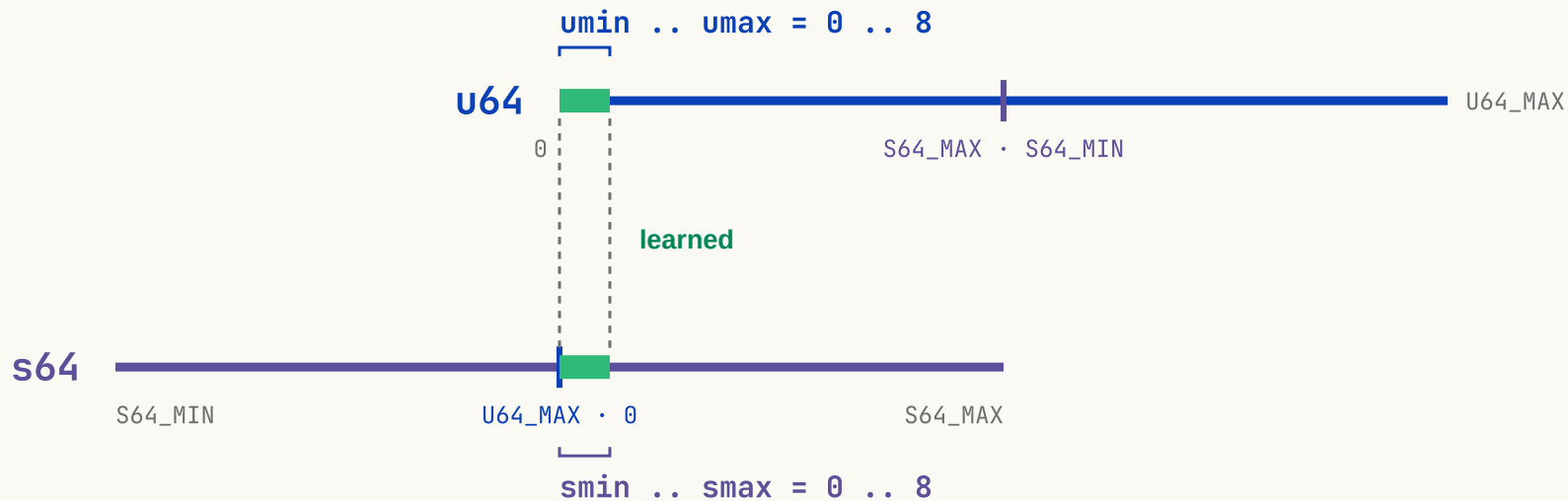
smax = 1



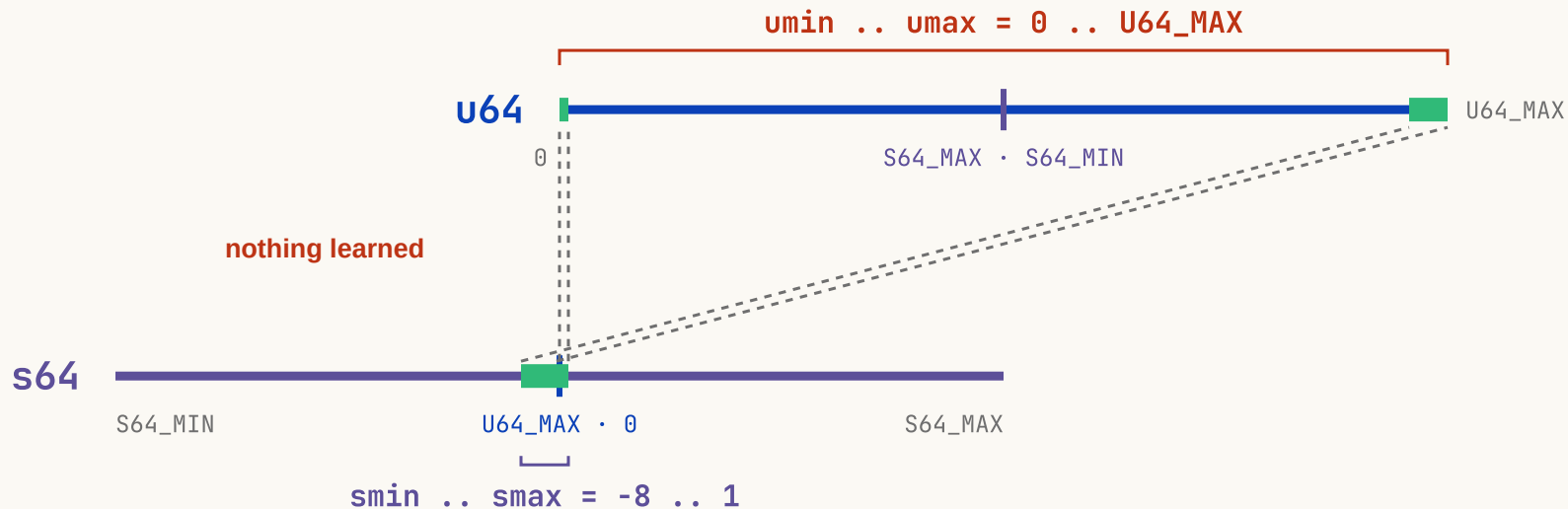
same program. the u64 line still says anything, so -8 is caught. rejected.



the unsigned line knows 0 .. 8. the signed line still says anything.



unsigned 0 .. 8 stays below the sign bit. so signed is 0 .. 8 too.



signed $-8 \dots 1$ crosses zero. as unsigned it splits in two, and one range has to take everything.

What about bits?

ABSTRACT: ONE TNUM

CONCRETE VALUES



0xa0 ... 0xaf

16 values



0xf0

a constant



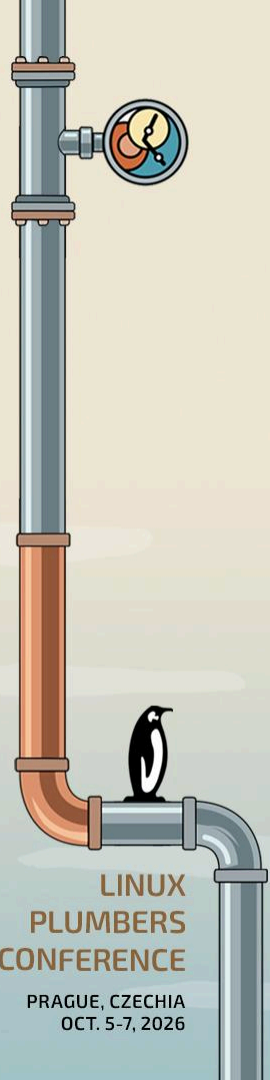
0xa0

exactly one value

exactly 0xa0.

tnum

More on this: [Making Sense of Tristate Numbers](#), LPC 2024



V5.7

2020, again

3f50f132d840 · John Fastabend

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

```
int len = ...;          /* int, not size_t */
```

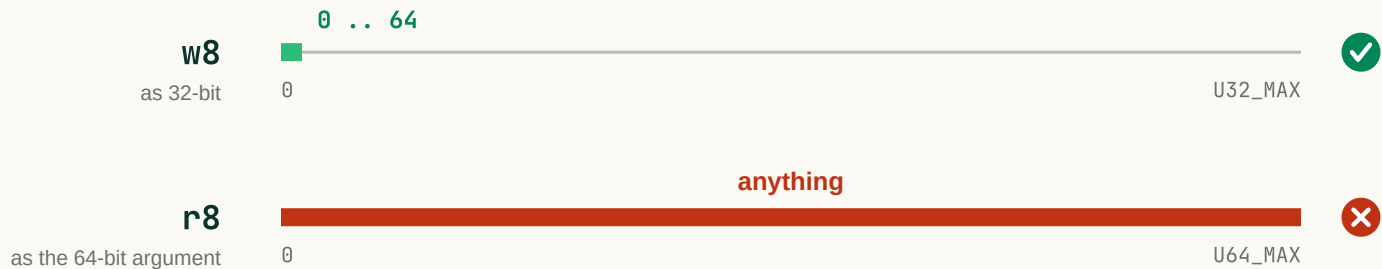
```
if (len < 0 || len > 64)  
    return 0;
```

```
bpf_probe_read(buf, len, src);
```

PROGRAM	R8, AS THE VERIFIER SEES IT	VERIFIER STORES
r8 = len from memory: anything	 0 U64_MAX	unknown
if w8 s< 0 goto out 32-bit compare: only the low half	 0 U64_MAX	unchanged
if w8 s> 64 goto out w8 is 0 .. 64. r8 is not.	 0 U64_MAX	unchanged
r2 = r8; call bpf_probe_read r2: 0 .. U64_MAX	 0 U64_MAX	umax = U64_MAX 

the checks bound w8. the verifier only tracks r8. nothing learned, so it is rejected.

r8 64 bits



the bottom half is bounded. the register is not.

Track the lower 32 bits too

1

imm

2014 · v3.18

2

max_value
min_value

2016 · v4.9

6

var_off.mask
var_off.value
umax_value
umin_value
smax_value
smin_value

2017 · v4.14

10

u32_max_value
u32_min_value
s32_max_value
s32_min_value
var_off.mask
var_off.value
umax_value
umin_value
smax_value
smin_value

2020 · v5.7

8 bounds, 10 numbers with the tnum. for one register.

3f50f132d840 · struct bpf_reg_state

**And they all have
to agree**

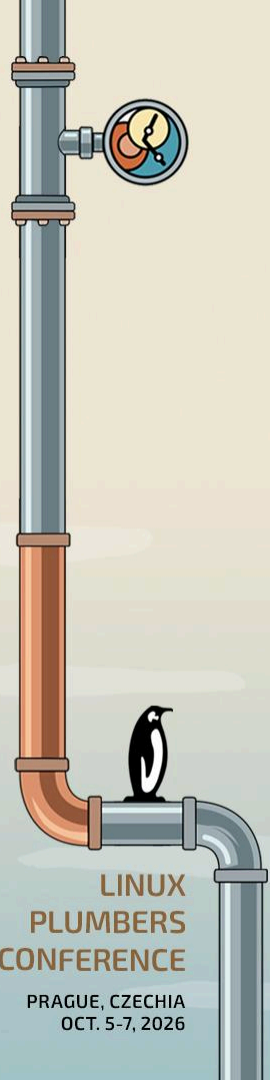
```
reg_bounds_sync(reg)
```

```
    __update_reg_bounds(reg);  
    __reg_deduce_bounds(reg);  
    __reg_deduce_bounds(reg);  
    /* yes, twice */  
    __reg_bound_offset(reg);  
    __update_reg_bounds(reg);
```

<code>deduce_bounds_64_from_64()</code>	136
<code>deduce_bounds_32_from_64()</code>	77
<code>deduce_bounds_32_from_32()</code>	48
<code>deduce_bounds_64_from_32()</code>	66

327 lines, just to keep 8 numbers consistent.

bbc631085503^ · line counts from the tree



SECTION 4

cnum

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



V7.2

2026

bbc631085503 • Eduard Zingerman

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

0

U64_MAX



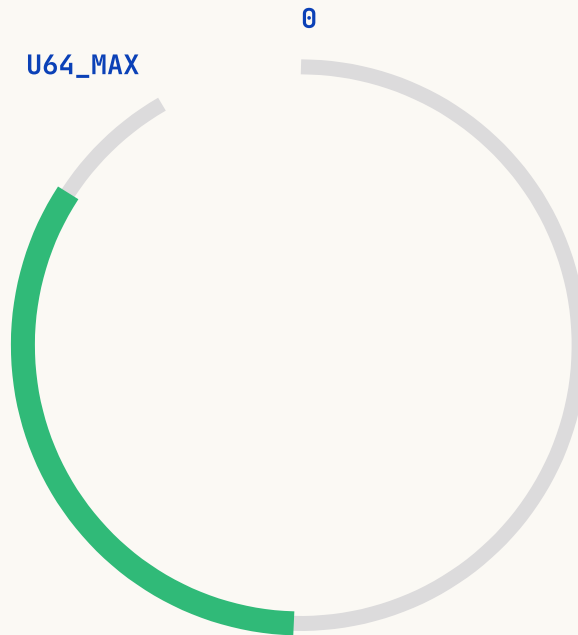
0

U64_MAX

0

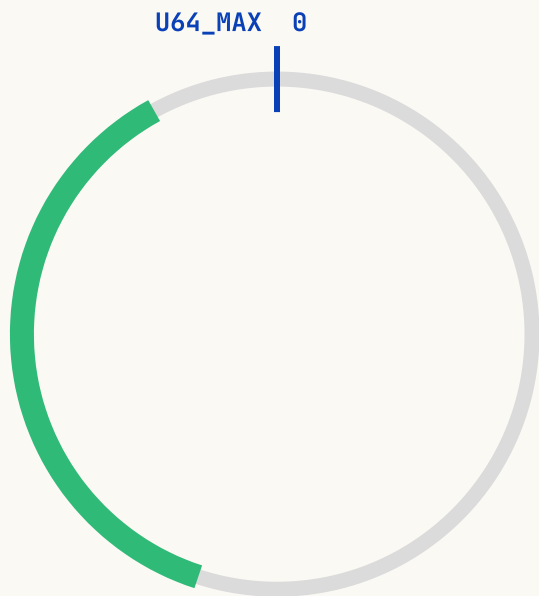
U64_MAX



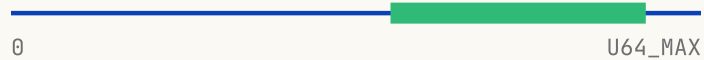


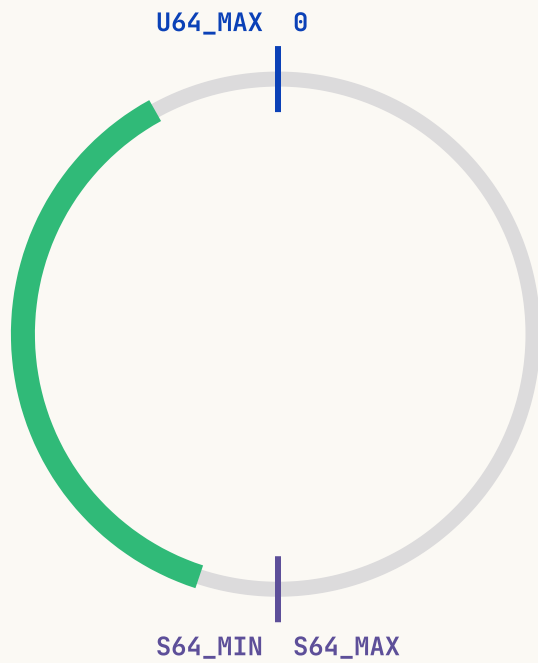
U64_MAX 0



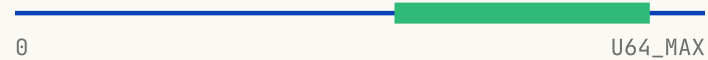


unsigned: cut at U64_MAX | 0



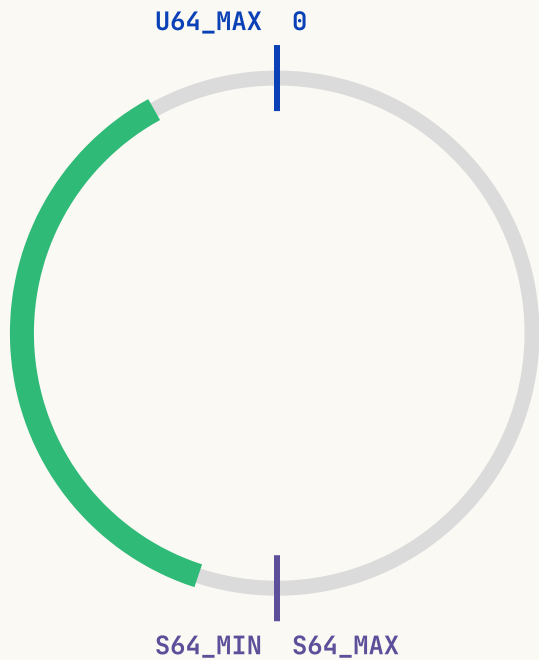


unsigned: cut at U64_MAX | 0



signed: cut at S64_MAX | S64_MIN





unsigned: cut at U64_MAX | 0



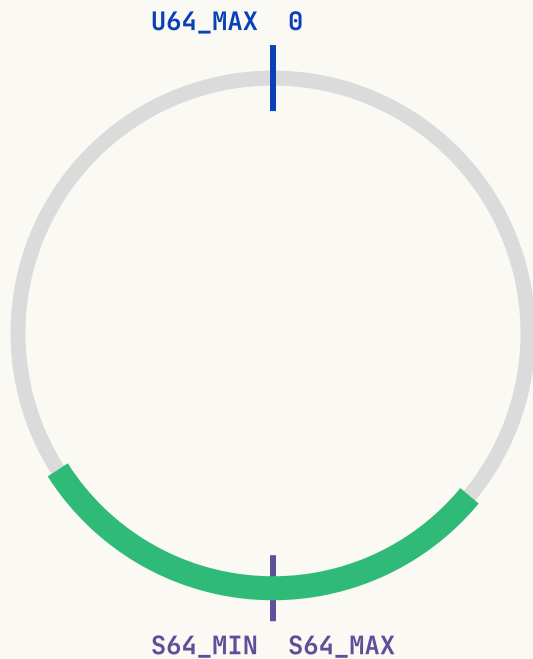
✓ one range

signed: cut at S64_MAX | S64_MIN

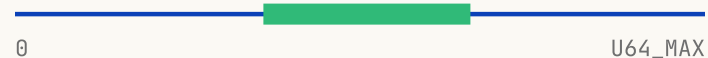


✓ one range

both readings work



unsigned: cut at U64_MAX | 0



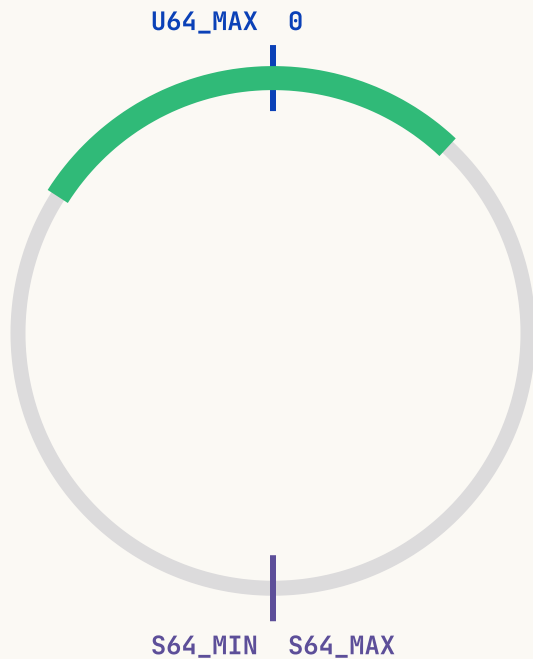
✓ one range

signed: cut at S64_MAX | S64_MIN

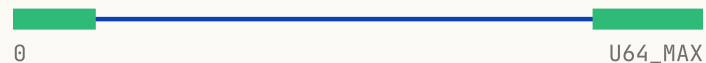


✗ two pieces: widen to everything

unsigned works, signed does not



unsigned: cut at U64_MAX | 0



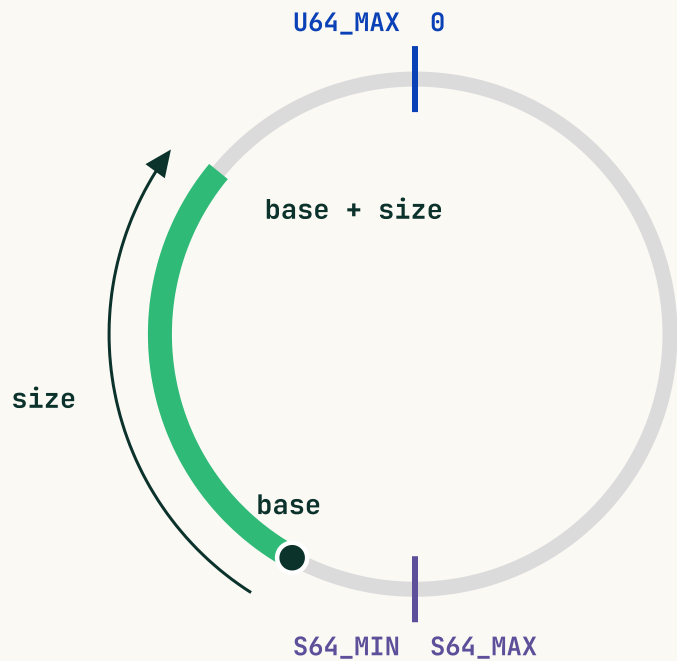
✗ two pieces: widen to everything

signed: cut at S64_MAX | S64_MIN



✓ one range

unsigned does not work, signed does



base

where the arc starts

size

how far it goes, clockwise

10

u32_max_value
u32_min_value
s32_max_value
s32_min_value
var_off.mask
var_off.value
umax_value
umin_value
smax_value
smin_value

2020 · v5.7



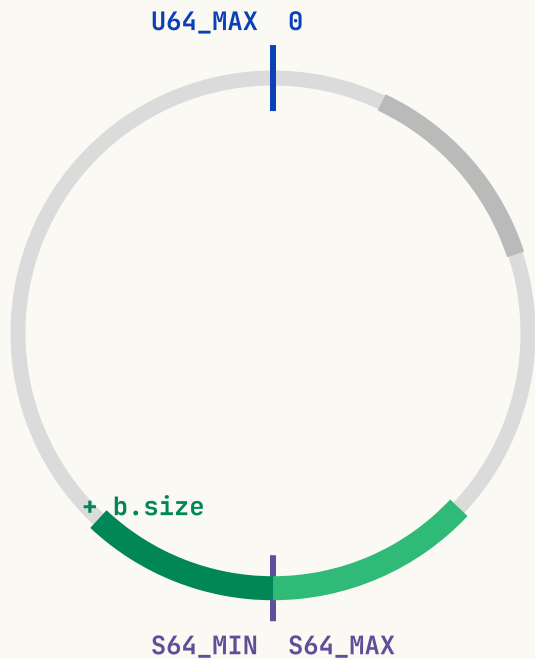
6

r32.size
r32.base
var_off.mask
var_off.value
r64.size
r64.base

2026 · v7.2

8 bounds become 2 arcs. 10 numbers down to 6.

bbc631085503 · struct bpf_reg_state



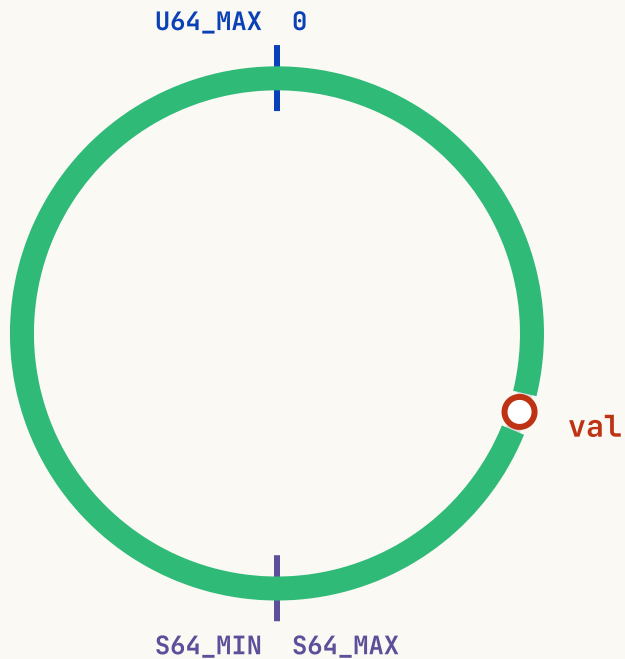
`c = a + b`

`c.base = a.base + b.base`

`c.size = a.size + b.size`

widen it by `b.size`. that is the whole operation.

**No overflow case.
Nowhere to put one.**



`r1 != val`

one arc: starts at `val + 1`,
goes all the way round to `val - 1`

✗ a range can only shave the ends

and `r1 != val`: everything except one value. still one arc.

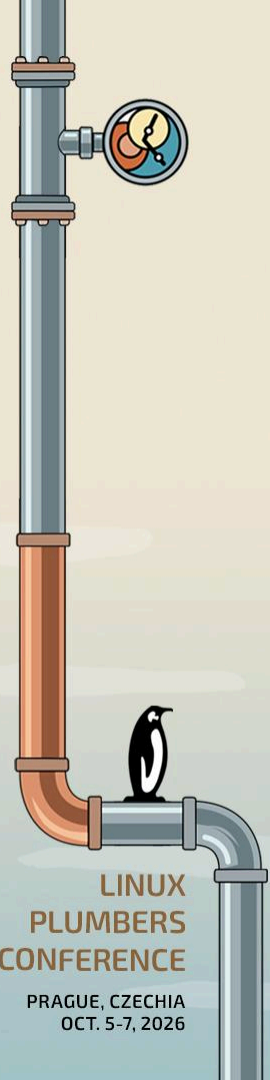
an interval cannot hold this at all

327 → 2

lines of bounds deduction

```
cnum64_add()      proved  
cnum64_negate()   proved  
cnum64_intersect() proved  
cnum64_is_subset() proved
```

checked with CBMC before the series was merged.



SECTION 5

Beyond Values

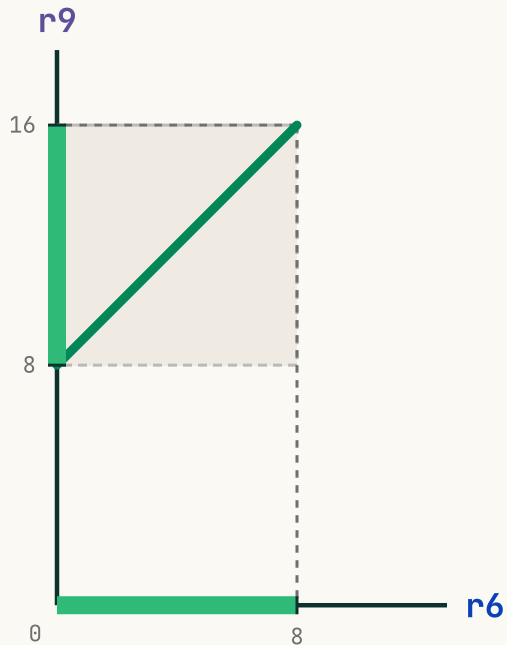
LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

**Not what one register holds.
How two registers relate.**

```
2020    r9 = r6           // same value
2024    r9 = r6 + 8       // same value, plus 8
2026    w7 = w6           // same low 32 bits
```

on the list now.



$r6 = 0 \dots 8$

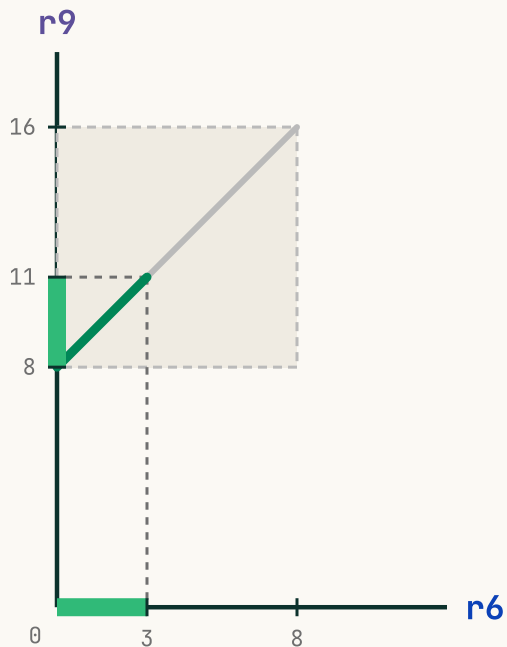
$r9 = r6 + 8$

 two number lines alone: the whole box

 with the relation: one line

 its shadow on each axis:
 $r6 \ 0 \dots 8, \ r9 \ 8 \dots 16$

track the line. each register's range is its shadow on an axis.



if r6 > 3 goto out

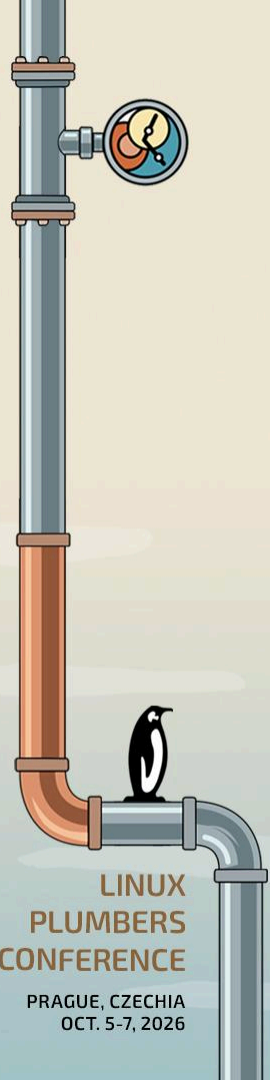
fall-through: $r6 \leq 3$

 the line is cut at $r6 = 3$

 both shadows shrink:
r6 0 .. 3, r9 8 .. 11

 one line each: r9 stays 8 .. 16

check r6, and r9 narrows with it. the line carries the knowledge across.

A vertical pipe with a pressure gauge and a penguin perched on a bend. The pipe is grey with orange-brown sections. A pressure gauge is attached to the top. A penguin is perched on a bend in the pipe. The background is a soft-focus landscape with a yellow sky and blue water.

SECTION 6

Future

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Put CBMC in BPF selftests

Or don't track at all

userspace

computes the invariants, emits a certificate

kernel

checks the program against it

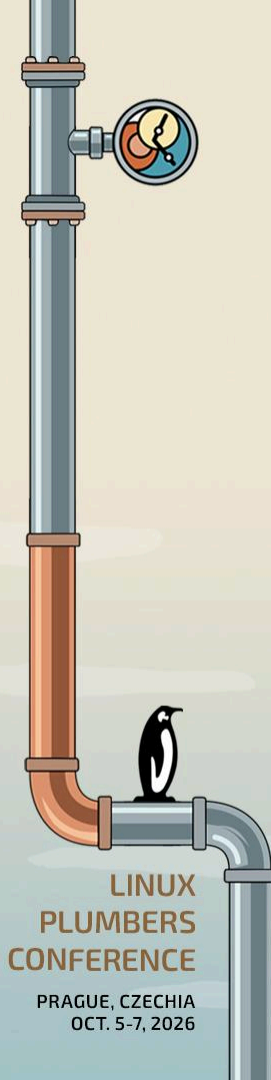
proof-carrying code. the analysis leaves the kernel.

Thank you

`kernel/bpf/cnum_defs.h`
`include/linux/cnum.h`

`github.com/eddyz87/cnum-verif`
`github.com/bpfverif/agni`

Shung-Hsi Yu • SUSE



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026