The ISoLA 2012 symposium
Heraklion, Crete
15 Oct 2012

LinuxTesting.org

# Pointer analysis with uninterpreted functions

Mikhail Mandrykin
mandrykin@ispras.ru

ISP RAS
Institute for System Programming of the Russian Academy of Sciences

# Linux Driver Verification

Currently a model can be accurate if it ***doesn't*** make heavy use of (or rely on):

- Bitwise operations (e.g. &, |, ^)
- Arithmetic over/underflow error detection
- Non-linear arithmetic
- Multithreading (race conditions detection)
- <u>Recursive data types (e.g. lists)</u>
- <u>Arrays</u>
- <u>Nested structures</u> (and `container_of` macro)
- <u>Pointer aliasing</u>

# Linux Driver Verification (2)

## Bugfix example

(/drivers/connector/connector.c, commit 663dd6d, Linux 2.6.40 i.e. also 3.0):

```
static int cn_call_callback(struct sk_buff *skb)
{
    int err = -ENODEV;
    ...
    if (cbq != NULL) {
        err = 0;
        ...
        kfree_skb(skb);
    }
    return err;
}
...
    err = cn_call_callback(skb);
    if (err < 0)
        kfree_skb(skb);
```

Double freeing of a socket buffer (skb)

# Linux Driver Verification (3)

Typical example of rule with *inaccurate* model:

- Don't call `kfree_skb` twice
  - Avoid double freeing of skb buffers

- Socket buffers (or skbs) are represented by `sk_buff` structures
- Pointers to the structures are frequently located in <u>arrays</u>, <u>lists</u>, <u>queues</u> and other data structures
- This leads to many *false positive* (spurious UNSAFE) verdicts

# Pointer analysis

BLAST with Andersen's alias analysis

- In many cases it's really useful!
  - ```
    kfree_skb(info->rx_skb);
    ```
  - ```
    info->rx_skb = NULL;
    ...
    dtl1_receive(info);
    ```
  - ```
    dtl1_receive(dtl1_info_t *info)
    {
      ...
      if (info->rx_skb)
        kfree_skb(info->rx_skb);
    ```

The infos here are *may-aliased*

The call is unreachable
    ```

# Pointer analysis (2)

But still...

- ```
  for (i = 0; i < RIONET_RX_RING_SIZE; i++)
      kfree_skb(rnet->rx_skb[i]);
  ```
  Arrays
  ```
  // RIONET_RX_RING_SIZE defaults to 128
  ```
- ```
  while (db->rx_avail_cnt) {
      kfree_skb(db->rx_ready_ptr->rx_skb_ptr);
      db->rx_ready_ptr = db->rx_ready_ptr->next_rx_desc;
      db->rx_avail_cnt--;
   }
  ```
  Linked lists
  ```
  // db->rx_avail_cnt <= RX_DESC_CNT == 32
  ```
- ```
  nf_conntrack_put_reasm(skb->nfct_reasm);
  ```
  Field to variable aliasing
  ```

# Pointer analysis (3)

But still...

- ```
  static void bdx_tx_free_skbs(struct bdx_priv *priv)
  {
      struct txdb *db = &priv->txdb;
      while (db->rptr != db->wptr) {
          kfree_skb(db->rptr->addr.skb);
          ++db->rptr;
      }
  }
  ```
  Pointer arithmetic
- **`struct`** `bdx_priv` has *26* fields. *11* are other structures or pointers to structures, some of which have more than *25* fields, some of which are also structures... and we are to update every subfield in each of `priv` may-aliases when passing the parameter
  ```

# Pointer analysis (4)

## BLAST's and alternative approaches

| Tool/approach | Pointers | Structures | Arrays | Recursive data structures | Pointer arithmetic | Performance |
|---|---|---|---|---|---|---|
| **BLAST** with *"closure depth"* | + | ± (finite depth) | - | - | - | + |
| Optimized **BLAST** with *"infinite closure depth"* | + | + | - | - | - | + |
| **BLAST** with *lazy shape analysis ("BLAST 3.0")* | + | + | - | + | - | ? |
| Bounded Model Checking | + | + | + | ± (finite depth) | + | - |
| **CPAchecker** with predicate analysis(current implementation) | + | - | - | - | - | + |
| *Our approach (uninterpreted functions)* | + | + | + | ± (finite depth) | + | ? |

# Suggested approach

What is the *initial* idea behind the approach?

- Very ***simple***, terribly *inefficient* but ***precise*** memory model:

`rnet->rx_skb[i]->users--;` →

$$M2 = store(M1,\ M1[M1[\mathbf{rnet}] + offsetof(\mathtt{struct\ rionet\_private}, \mathbf{rx\_skb}) + M1[\underline{i}]] + offsetof(\mathtt{struct\ sk\_buff}, \mathbf{users}),\ M1[M1[M1[\mathbf{rnet}] + offsetof(\mathtt{struct\ rionet\_private}, \mathbf{rx\_skb}) + M1[\underline{i}]] + offsetof(\mathtt{struct\ sk\_buff}, \mathbf{users})] - 1)$$

# Suggested approach (2)

Even more inefficient...

- Most state-of-the-art SMT-solvers still don't fully support array interpolation
- So we'll use ***uninterpreted functions***
- **No** *store(·,·,·)* operation
- Congruence: $a = b \rightarrow f(a) = f(b)$
- $m_1(a_1) = 1$, $m_2(a_2) = 2$, $m_2(a_1) = ?$
- We need to explicitly encode retention of earlier assigned values
- $a_2 \neq a_1 \rightarrow m_2(a_1) = m_1(a_1)$ and so on for every $a_i$

# Suggested approach (3)

How do we encode memory regions?

- One uninterpreted constant for each region
- Each region has positive address *($b_i > 0$)*
- Regions don't intersect:
  $B(b_i + k) = i,\ 0 \le k < s,$
  where *s* is the size of the region
- $b_i + k = b_j + l \rightarrow B(b_i + k) = B(b_j + l) \rightarrow i = j$
- So number of such equalities is linear

# Suggested approach (4)

What are suggested optimizations?

- ***Typing*** i.e. one array per one simple data type

e.g. **`char`** `*`, **`long int`**, **`struct`** `sk_buff *`, `...`

- ***Pure variables***, i.e. variables that don't have aliases

e.g. **`int`** `i;` // just a counter
// \``&i`' occurs **nowhere** in the code

- ***Structure field*** assignment ***optimization***, i.e. omitting the antecedents if offsets are known to be unequal in advance

e.g. updating `skb1->next` *can't* influence *any* `skb2->prev` though they have the same type

# Suggested approach (5)

Further optimizatinos

- Using ***constant subexpressions*** for initialization

e.g. `kzalloc(sizeof(*info), GFP_KERNEL)`

- ***Amortization*** of sequential assignments

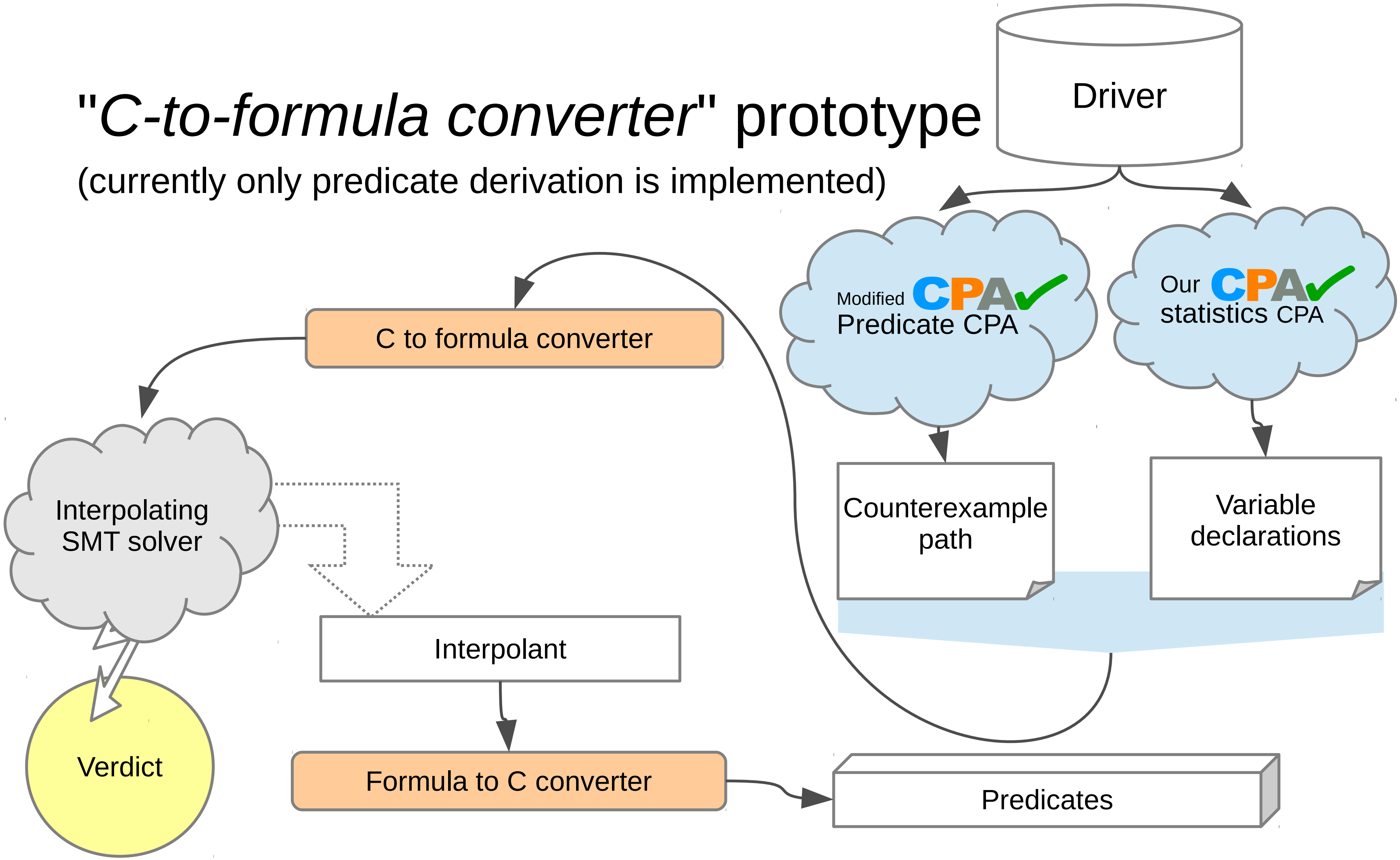
e.g.
```
for (i = 0; i < MAX_SKB_FRAGS + 1; i++) {
    lwords = 7 + (i * 3);
    ... /* pad it with 1 lword */
    txd_sizes[i].qwords = lwords >> 1;
    txd_sizes[i].bytes = lwords << 2;
}
// No reading through any pointer during the entire loop
// So let's update the memory just once after the loop!
```

- Applying preliminary alias analysis (again!)

# Evaluation

"*C-to-formula converter*" prototype

(currently only predicate derivation is implemented)



# Evaluation (1)

**Pointer target statistics**

| Driver | Total | By type | | | | | **By offset** (max) |
|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | |
| bluetooth/bpa10x.ko (2 skb) | 422 | `unsigned char` 134 | `signed char` 112 | `unsigned long int` 44 | `unsigned short int` 26 | `signed long int` 18 | 18 |
| bluetooth/dtl1_cs.ko (32 skb) | 3814 | `signed char` 1609 | `unsigned char` 639 | `unsigned long int` 536 | `unsigned short int` 312 | `signed long int` 170 | 137 |
| isdn/hysdn/hysdn.ko (20 skb) | 2120 | `signed char` 960 | `unsigned char` 340 | `unsigned long int` 300 | `unsigned short int` 180 | `signed long int` 100 | 80 |
| hid/usbhid/usbkbd.ko (no rule model applied) | 789 | `unsigned char` 293 | `signed char` 329 | `unsigned long int` 39 | `struct list_head*` 12 | `unsigned short int` 12 | 11 |
| net/usb/cdc-phonet.ko (no rule model applied) | 224 | `unsigned long int` 38 | `signed char` 59 | `signed char *` 16 | `unsigned char` 16 | `struct list_head*` 12 | 17 |

# Evaluation (2)

**Formula sizes in KB**

| Driver | No optimizations | Target filtering | Pure variables | Both |
|---|---|---|---|---|
| bluetooth/bpa10x.ko (2 skb) | 448 | 369 | 311 | 220 |
| bluetooth/dtl1_cs.ko (32 skb) | 3700 | 2400 | 623 | 456 |
| isdn/hysdn/hysdn.ko (20 skb) | 726 | 474 | 101 | 95 |
| hid/usbhid/usbkbd.ko (no rule model applied) | 352 | 279 | 119 | 88 |
| net/usb/cdc-phonet.ko (no rule model applied) | 255 | 166 | 15 | 15 |

# Evaluation (3)

## Sample interpolant and predicates

```
true
( and
  (= usbpn_open!!i~1 0.0)
  (= usbpn_open!!dev~1 usbpn_open!!pnd~1))
( and
  (= usbpn_open!!i~1 0.0)
  (= usbpn_open!!dev~1 usbpn_open!!pnd~1))
(= (struct-urb-*~2 (+ (+ usbpn_close!!pnd~1 usbpn_close!!i~1) 66.0)) 0.0)
false
```

~70 locations
~60 operators in path
MathSAT interpolation time: 0.039s

```
0 == 0
usbpn_open::i == 0 && usbpn_open::dev == usbpn_open::pnd
usbpn_open::i == 0 && usbpn_open::dev == usbpn_open::pnd
usbpn_close::pnd->urbs[usbpn_close::i] == 0
0 < 0
```

# Conclusions

- Approach isn't carefully evaluated yet
- Current results are not disappointing
- Not all optimizations are implemented → better performance expected
- Many low-level C language features are supported (such as pointer arithmetics, `container_of` macro etc.)
- We are planning further investigation of the approach and its implementation as a CPA in the CPAchecker(CPA) tool

# Thank you!

Mikhail Mandrykin
mandrykin@ispras.ru

Institute for System Programming of the Russian Academy of Sciences