Pointer analysis with uninterpreted functions

Mikhail Mandrykin
mandrykin@ispras.ru
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)
- Recursive data types (e.g. lists)
- Arrays
- Nested structures (and container_of macro)
- Pointer aliasing
Bugfix example

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

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

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 arrays, lists, queues 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);`
But still...

- **for** ($i = 0; i < RIONET_RX_RING_SIZE; i++)
  
  ```c
  kfree_skb(rnet->rx_skb[i]);
  ```

  // RIONET_RX_RING_SIZE defaults to 128

- **while** (db->rx_avail_cnt) {
  
  ```c
  kfree_skb(db->rx_ready_ptr->rx_skb_ptr);
  db->rx_ready_ptr = db->rx_ready_ptr->next_rx_desc;
  db->rx_avail_cnt--;
  ```

  // db->rx_avail_cnt <= RX_DESC_CNT == 32

- **nf_conntrack_put_reasm(skb->nfct_reasm);**
But still...

```c
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;
    }
}
```

- **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

<table>
<thead>
<tr>
<th>Tool/approach</th>
<th>Pointers</th>
<th>Structures</th>
<th>Arrays</th>
<th>Recursive data structures</th>
<th>Pointer arithmetic</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BLAST</strong> with “closure depth”</td>
<td>+</td>
<td>±</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td><strong>Optimized BLAST</strong> with “infinite closure depth”</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td><strong>BLAST</strong> with lazy shape analysis (“BLAST 3.0”)</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>?</td>
</tr>
<tr>
<td>Bounded Model Checking</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>±</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td><strong>CPAchecker</strong> with predicate analysis(current implementation)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td><strong>Our approach</strong> (uninterpreted functions)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>±</td>
<td>+</td>
<td>?</td>
</tr>
</tbody>
</table>
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[rnet] + offsetof(struct rionet_private, rx_skb) + M1[i]] +
offsetof(struct sk_buff, users),
M1[M1[M1[rnet] + offsetof(struct rionet_private, rx_skb) + M1[i]] +
offsetof(struct sk_buff, 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** \( \text{store}(\cdot,\cdot,\cdot) \) 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:
  \[
  B(b_i + k) = i, \ 0 \leq k < s,
  \]
  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

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

- **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

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

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

Suggested approach (5)

Further optimizations

- **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

<table>
<thead>
<tr>
<th>Driver</th>
<th>Total</th>
<th>By type</th>
<th>By offset (max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bluetooth/bpa10x.ko (2 skb)</td>
<td>422</td>
<td>unsigned char&lt;br&gt;134</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed char&lt;br&gt;112</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned long int&lt;br&gt;44</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned short int&lt;br&gt;26</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed long int&lt;br&gt;18</td>
<td></td>
</tr>
<tr>
<td>bluetooth/dtl1_cs.ko (32 skb)</td>
<td>3814</td>
<td>signed char&lt;br&gt;1609</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned char&lt;br&gt;639</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned long int&lt;br&gt;536</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned short int&lt;br&gt;312</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed long int&lt;br&gt;170</td>
<td></td>
</tr>
<tr>
<td>isdn/hysdn/hysdn.ko (20 skb)</td>
<td>2120</td>
<td>signed char&lt;br&gt;960</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned char&lt;br&gt;340</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned long int&lt;br&gt;300</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned short int&lt;br&gt;180</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed long int&lt;br&gt;100</td>
<td></td>
</tr>
<tr>
<td>hid/usbhid/usbkbd.ko (no rule model applied)</td>
<td>789</td>
<td>unsigned char&lt;br&gt;293</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed char&lt;br&gt;329</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned long int&lt;br&gt;39</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>struct list_head*&lt;br&gt;12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned short int&lt;br&gt;12</td>
<td></td>
</tr>
<tr>
<td>net/usb/cdc-phonet.ko (no rule model applied)</td>
<td>224</td>
<td>unsigned long int&lt;br&gt;38</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed char&lt;br&gt;59</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed char&lt;br&gt;16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>unsigned char&lt;br&gt;16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>struct list_head*&lt;br&gt;12</td>
<td></td>
</tr>
</tbody>
</table>
### Evaluation (2)

**Formula sizes in KB**

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

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

~70 locations
~60 operators in path
MathSAT interpolation time: 0.039s
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 tool
Thank you!

Mikhail Mandrykin
mandrykin@isprras.ru