Part I. Hunting for Bugs

Vadim Mutilin

Institute for System Programming of the Russian Academy of Sciences
**Problems in Linux Kernel**

This section contains information about problems in Linux kernel found within Linux Driver Verification program as well as within KEDR and Linux File System Verification projects.

Click on a problem number for detailed description. Click on a column header to change the sorting order.

<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Brief</th>
<th>Added on</th>
<th>Accepted</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0351</td>
<td>Crash</td>
<td>regulator: tps65217: NULL pointer dereference on probe</td>
<td>2019-09-26</td>
<td></td>
<td>Fixed in kernel v4.19-rc1</td>
</tr>
<tr>
<td>L0350</td>
<td>Crash</td>
<td>scsi: 3ware: fix return 0 on the error path of probe</td>
<td>2019-09-25</td>
<td></td>
<td>Fixed in kernel v4.19-rc1</td>
</tr>
<tr>
<td>L0349</td>
<td>Crash</td>
<td>gpio: ml-ioh: buffer underwrite on probe error path</td>
<td>2019-09-24</td>
<td></td>
<td>Fixed in kernel v4.19-rc1</td>
</tr>
<tr>
<td>L0348</td>
<td>Crash</td>
<td>firmware: vpd: incorrect section enabled flag on vpd_section_destroy</td>
<td>2019-09-20</td>
<td></td>
<td>Fixed in kernel v4.19-rc1</td>
</tr>
<tr>
<td>L0346</td>
<td>Leak</td>
<td>media: dw2102: memleak in dw2102_probe</td>
<td>2019-09-20</td>
<td></td>
<td>Fixed in kernel v4.19-rc1</td>
</tr>
<tr>
<td>L0345</td>
<td>Leak</td>
<td>uwb: hwa-rc: memory leak at hwarc_probe</td>
<td>2019-09-20</td>
<td></td>
<td>Fixed in kernel v4.19-rc1</td>
</tr>
<tr>
<td>L0344</td>
<td>Crash</td>
<td>tty: rocket: possible buffer overwrite on register_PCI</td>
<td>2019-09-20</td>
<td></td>
<td>Fixed in kernel v4.19-rc1</td>
</tr>
</tbody>
</table>
Bugs Found for Subsystems
Bugs Found for Subsystems
Problems in Linux Kernel Found by CPAchecker

This section contains information about problems in Linux kernel found within Linux Driver Verification project with CPAchecker engine.

Click on a problem number for detailed description. Click on a column header to change the sorting order.

There are 158 bugs found by CPAchecker.

<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Brief</th>
<th>Added on</th>
<th>Accepted</th>
<th>Status</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0350</td>
<td>Crash</td>
<td>scsi: 3ware: fix return 0 on the error path of probe</td>
<td>2019-09-25</td>
<td><a href="https://lkml.org/lkml/2018/7/27/655">https://lkml.org/lkml/2018/7/27/655</a> commit</td>
<td>Fixed in kernel v4.19-rc1</td>
<td>...</td>
</tr>
<tr>
<td>L0348</td>
<td>Crash</td>
<td>firmware: vpd: incorrect section enabled flag on vpd_section_destroy</td>
<td>2019-09-20</td>
<td><a href="https://lkml.org/lkml/2018/7/23/944">https://lkml.org/lkml/2018/7/23/944</a> commit</td>
<td>Fixed in kernel v4.19-rc1</td>
<td>...</td>
</tr>
</tbody>
</table>
Bugs Found by CPAchecker
Total Bugs Found
Top 10 of 35 Rules

- Clock: 15%
- Mutex: 13%
- USB device: 11%
- DMA: 6%
- Memsafe: 7%
- URB: 6%
- Memory leak: 3%
- Spinlock: 4%
- Races: 5%
- Might sleep: 6%
Consequences

- Leak, 38.87%
- Use of uninitialized variable, 14.54%
- Race, 11.28%
- Deadlock, 10.39%
- Use after free, 7.42%
- Sleep in atomic, 5.34%
- NPD, 3.26%
- Buffer overrun, 1.19%
- Others, 7.72%
Consequences (by the tool)
On the Error Path? (by the tool)
On the Error Path? (for top 10 rules)
Part II. ARINC to AADL

Sergey Lesovoy

Institute for System Programming of the Russian Academy of Sciences
ARINC 653

From Wikipedia, the free encyclopedia

ARINC 653 (Avionics Application Standard Software Interface) is a software specification for space and time partitioning in safety-critical avionics real-time operating systems (RTOS). It allows the hosting of multiple applications of different software levels on the same hardware in the context of an Integrated Modular Avionics architecture.[1]

- Partition & process management

```c
process_attrs.ENTRY_POINT = first_process;
strncpy(process_attrs.NAME, "process 1", sizeof(PROCESS_NAME_TYPE));

CREATE_PROCESS(&processAttrs, &pid, &ret);

SET_PARTITION_MODE(NORMAL, &ret);
```

- Inter & intra partition communication

```c
// create ports
CREATE_QUEUEING_PORT("QP1", 64, 10, DESTINATION, FIFO, &QP1, &ret);

MESSAGE_SIZE_TYPE len;
RECEIVE_QUEUEING_MESSAGE(QP1, INFINITE_TIME_VALUE, (MESSAGE_ADDR_TYPE *)&msg, &len, &ret);
```

- etc
Architecture Analysis and Design Language (AADL)

Example 1.
Communicating with intra partition port and global variable

Example 2.
Communicating with inter partition port
ARINC processes

“process 1” does not start here

```
process_attrs.ENTRY_POINT = first_process;
strncpy(process_attrs.NAME, "process 1", sizeof(PROCESS_NAME_TYPE));
CREATE_PROCESS(&process_attrs, &pid, &ret);

// create ports
CREATE_QUEUEING_PORT("QP1", 64, 10, DESTINATION, FIFO, &QP1, &ret);

SET_PARTITION_MODE(NORMAL, &ret);
```

Initialization of ARINC entities

“process 1“ starts here
ARINC processes
Solution 1

- Preliminary Value analysis collecting set of process function pointers
- Inserting function calls explicitly

```c
process attrs.ENTRY_POINT = first_process;
strncpy(process_attrs.NAME, "process 1", sizeof(PROCESS_NAME_TYPE));
CREATE_PROCESS(&process_attrs, &pid, &ret);

// create ports
CREATE_QUEUEING_PORT("QP1", 64, 10, DESTINATION, FIFO, &QP1, &ret);
SET_PARTITION_MODE(NORMAL, &ret);

first_process();
```

Start “process 1”
ARINC processes
Solution 2

- A model with nondeterministic choice

```c
SYSTEM_ENTRY_TYPE nondet_entry_point = 0;

void CREATE_PROCESS (  
    PROCESS_ATTRIBUTE_TYPE *attributes,  
    PROCESS_ID_TYPE *process_id,  
    RETURN_CODE_TYPE *return_code)
{
    if(__VERIFIER_nondet_int()) {
        nondet_entry_point = attributes->ENTRY_POINT;
    }
}

void SET_PARTITION_MODE (...)   
{
    (*nondet_entry_point)();
}
```

Save nondeterministically

Call saved pointer
ARINC entities

Creation of process with name “process 1” and function pointer first_process. Identifier is stored in variable pid.

Creation of port entity with name “QP1”. Identifier is stored in variable QP1.

Entering “process 1”
ARINC entities
Solution

pid → {“process 1”, first_function}

process_attrs.ENTRY_POINT = first_function;
strncpy(process_attrs.NAME, “process 1”, sizeof(PROCESS_NAME_TYPE));
CREATE_PROCESS(&process_attrs, &pid, &ret);
// create ports
CREATE_QUEUEING_PORT("QP1", 64, 10, DESTINATION, FIFO, &QP1, &ret);
SET_PARTITION_MODE(NORMAL, &ret);

void first_process() {
...
MESSAGE_SIZE_TYPE len;
RECEIVE_QUEUEING_MESSAGE(QP1, INFINITE_TIME_VALUE, (MESSAGE_ADDR_TYPE) &msg, &len, &ret);
ARINC entities
Solution

pid → {“process 1”, first_function}

```c
process_attrs.ENTRY_POINT = first_process;
strncpy(process_attrs.NAME, "process 1", sizeof(PROCESS attrs));
CREATE_PROCESS(&process_attrs, &pid, &ret);

// create ports
CREATE_QUEUEING_PORT("QP1", 64, 10, DESTINATION, FIFO, &QP1, &ret);
SET_PARTITION_MODE(NORMAL, &ret);

void first_process() {
...
MESSAGE_SIZE_TYPE len;
RECEIVE_QUEUEING_MESSAGE(QP1, INFINITE_TIME_VALUE, (MESSAGE_ADDR_TYPE) &msg, &len, &ret);
```
ARINC entities
Solution

pid → {“process 1”, first_function}

QP1 → “QP1”

current → “process 1”

pid → {“process 1”, first_function}

QP1 → “QP1”
ARINC entities
Solution

pid → \{“process 1”, first_function\}

\begin{verbatim}
process_attrs.ENTRY_POINT = first_process;
strcpy(process_attrs.NAME, “process 1”, sizeof(PROCESS_)
CREATE_PROCESS(&process_attrs, &pid, &ret);

// create ports
CREATE_QUEUE_PORT("QP1", 64, 10, DESTINATION, FIFO, &QP1, &ret);
SET_PARTITION_MODE(NORMAL, &ret);

void first_process() {
...

MESSAGE_SIZE_TYPE len;
RECEIVE_QUEUE_MESSAGE(QP1, INFINITE_TIME_VALUE, (MESSAGE_ADDR_TYPE) &msg, &len, &ret);
\end{verbatim}

QP1 → “QP1”

current → “process 1”
pid → \{“process 1”, first_function\}
QP1 → “QP1”
Collect values only on reachable paths

1. The path should be reachable
2. Get the value
Collect values only on reachable paths

Solution. Refinement

```c
process_attrs.ENTRY_POINT = first_process;
strncpy(process_attrs.NAME, "process 1", sizeof(PROCESS_NAME_TYPE));

CREATE_PROCESS(&process_attrs, &pid, &ret);

// create ports
CREATE_QUEUEING_PORT("QP1", 64, 10, DESTINATION, FIFO, &QP1, &ret);
SET_PARTITION_MODE(NORMAL, &ret);

void first_process() {
...

    MESSAGE_SIZE_TYPE len;
    RECEIVE_QUEUEING_MESSAGE(QP1, INFINITE_TIME_VALUE, (MESSAGE_ADDR_TYPE *)&msg, &len, &ret);
```

Mark as target state (violation)
Take the value from Value analysis
CPAchecker ARINC2AADL

- ARINC processes
  - Solution 1. Preliminary analysis – requires modification of CFA
  - Solution 2. Nondeterministic choice – sound for sequential analysis only

- ARINC entities
  - Solution. Extension of Value analysis – supports pointers only heuristically
  - Generalize for other analyses?

- Collect values on reachable paths
  - Value Analysis – supports pointers only heuristically
  - Predicate Analysis – how to exclude undefined values?
Part III. Topics
Topics (unsorted)

- Collect data values with predicate analysis
- Correctness witness visualization
- Stepwise input program simplification and debugging of CPAchecker
- Type and BnB regions for array encoding in predicate analysis
- CPALockator
  - Support for atomic access primitives
  - Support for interrupts model
  - Shared analysis with refinement
  - Support for message passing
  - Support for control dependencies
- Checking memsafety properties for multithreaded programs
- Simplifying input source code for the verification (CIL-less)
- Loop iterations abstraction and refinement
- Generation of exploits
- Checking for undefined behavior with symbolic memory graphs
- Local path refinement selection in BAM
- On-demand memory for predicate analysis
- Runtime learning of environment models
Thank you!

Vadim Mutilin
http://linuxtesting.org/project/ldv