Linux Device-Drivers Verification Challenges

Alexey Khoroshilov
Vadim Mutilin
Eugene Novikov

Institute for System Programming of the Russian Academy of Sciences
Definitions to discuss
Static Analysis: Trade-Off Triangle

- False positives
- light-weight
- heavy-weight
- Time of analysis
- False negative
Static Analysis vs Model Checking

Static Analysis

Model Checking

potential bugs found

SAFE
UNSAFE
UNKNOWN

error trace
Linux Device Drivers

- Large and constantly increasing mass of
  - uniform
  - important
  - not too big
  - not too complex
- software code

6.5 MLOC
(*) drivers&sound in 3.6

+ 0.5 MLOC per year
(*) between 3.1 and 3.6

Drivers are similar enough

Share memory and privilege with the rest of the kernel

95% < 15 KLOC
90% < 10 KLOC
80% < 5 KLOC
(*) size of .ko

Source code available

No floating point, rarely use recursion
Linux Device Drivers

- Large and constantly increasing mass of software code
  - uniform
  - important
  - not too big
  - not too complex

- software code
  - Source code available

6.5 MLOC

(*) drivers & sound in 3.6

(*) between 3.1 and 3.6

95% < 15 KLOC

90% < 10 KLOC

80% < 5 KLOC

(*) size of .ko

Sounds like a dream
But...
But...

(1) No entry point
Model Checking World

- Reachability problem
Device Driver World

```c
static struct pci_driver DAC960_pci_driver = {
    .name = "DAC960",
    .id_table = DAC960_id_table,
    .probe = DAC960_Probe,
    .remove = DAC960_Remove,
};

static int DAC960_init_module(void)
{
    int ret;

    ret = pci_register_driver(&DAC960_pci_driver)
    #ifdef DAC960_GAM_MINOR
        if (!ret)
            DAC960_gam_init();
    #endif
    return ret;
}
...

module_init(DAC960_init_module);
module_exit(DAC960_cleanup_module);
```
Linux Kernel
(1) No entry point

- No predefined entry point
- Event handlers via function pointers
- Order limitation
  - open() after probe(), but before remove()
- Implicit limitations
  - read() only if open() succeed
- and the limitations are specific for each class of drivers
But...

(1) No entry point

(2) Kernel core model
Linux Kernel

Kernel-Device Interface

Interruptions, I/O ports, etc.

Kernel Core

Device Drivers

Kernel-Userspace Interfaces

Syscalls, procfs, etc.
(2) Kernel core model

- Boundary of driver
- Kernel core API properties
But...

(1) No entry point

(2) Kernel core model

(3) No stable API
stable_api_nonsense.txt

This is being written to try to explain why Linux does not have a binary kernel interface, nor does it have a stable kernel interface. Please realize that this article describes the in kernel interfaces, not the kernel to userspace interfaces. The kernel to userspace interface is the one that application programs use, the syscall interface. That interface is very stable over time, and will not break.

Executive Summary
You think you want a stable kernel interface, but you really do not, and you don't even know it. What you want is a stable running driver, and you get that only if your driver is in the main kernel tree. You also get lots of other good benefits if your driver is in the main kernel tree, all of which has made Linux into such a strong, stable, and mature operating system which is the reason you are using it in the first place.

Greg Kroah-Hartman
(3) No stable API

- Environment interface and invariants
- Kernel core model
But...

(1) No entry point

(2) Kernel core model

(3) No stable API

(4) Low level code
container_of

#define container_of(ptr, type, member) ({
const typeof( ((type *)0)→member ) * __mptr = (ptr); 
(type *)( (char *) __mptr − offsetof(type,member) );})

struct A {
    int f1;
    char[5] f2;
    struct B f3;
}

container_of(p, struct A, f3)
(4) Low level code

- pointer arithmetics
- casting
- container_of
But...

(1) No entry point
(2) Kernel core model
(3) No stable API
(4) Low level code
(5) Hardware specific
Linux Kernel
(5) Hardware specific

- Hardware specific bugs
- Hardware specific invariants
But...

(1) No entry point
(2) Kernel core model
(3) No stable API
(4) Low level code
(5) Hardware specific
(6) Concurrency
(6) Concurrency

- Device drivers are significantly asynchronous
- Many bugs appear in concurrent settings only
Commit Analysis

- All patches in stable trees (2.6.35 – 3.0) for 1 year:
- 1503 patches in device drivers
- Main goal: detect and classify typical bugs
# Commit Analysis (2)

<table>
<thead>
<tr>
<th>Class</th>
<th>#</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync:race</td>
<td>60</td>
<td>17.2%</td>
</tr>
<tr>
<td>specific:resource</td>
<td>32</td>
<td>9.2%</td>
</tr>
<tr>
<td>generic:null_ptr_deref</td>
<td>31</td>
<td>8.9%</td>
</tr>
<tr>
<td>specific:check_params</td>
<td>25</td>
<td>7.2%</td>
</tr>
<tr>
<td>generic:resource</td>
<td>24</td>
<td>6.9%</td>
</tr>
<tr>
<td>specific:context</td>
<td>19</td>
<td>5.4%</td>
</tr>
<tr>
<td>specific:uninit</td>
<td>17</td>
<td>4.9%</td>
</tr>
<tr>
<td>generic:syntax</td>
<td>14</td>
<td>4.0%</td>
</tr>
<tr>
<td>specific:lock</td>
<td>12</td>
<td>3.4%</td>
</tr>
<tr>
<td>sync:deadlock</td>
<td>11</td>
<td>3.2%</td>
</tr>
<tr>
<td>specific:style</td>
<td>10</td>
<td>2.9%</td>
</tr>
<tr>
<td>specific:net</td>
<td>10</td>
<td>2.9%</td>
</tr>
<tr>
<td>specific:usb</td>
<td>9</td>
<td>2.6%</td>
</tr>
<tr>
<td>generic:int_overflow</td>
<td>8</td>
<td>2.3%</td>
</tr>
<tr>
<td>generic:buffer_overflow</td>
<td>8</td>
<td>2.3%</td>
</tr>
<tr>
<td>specific:check_ret_val</td>
<td>7</td>
<td>2.0%</td>
</tr>
<tr>
<td>generic:uninit</td>
<td>6</td>
<td>1.7%</td>
</tr>
<tr>
<td>specific:dma</td>
<td>4</td>
<td>1.1%</td>
</tr>
<tr>
<td>specific:device</td>
<td>4</td>
<td>1.1%</td>
</tr>
<tr>
<td>specific:misc</td>
<td>27</td>
<td>7.7%</td>
</tr>
<tr>
<td>generic:misc</td>
<td>11</td>
<td>3.2%</td>
</tr>
</tbody>
</table>
Challenges

(1) No entry point

(2) Kernel core model

(3) No stable API

(4) Low level code

(5) Hardware specific

(6) Concurrency
Linux Verification Center

founded in 2005

- OLVER Program
- Linux Standard Base Infrastructure Program
- Linux Driver Verification Program
LDV Toolchain

- Linux Kernel
- Driver
- Rule Model

LDV Core
- Kernel Manager
- Build Cmd Extractor
- Driver Environment Generator

Domain Specific C Verifier
- Rule Instrumentor
- Reachability C Verifier

Verification engines
- Wrapper BLAST
- Wrapper CPAchecker

Verdict

Report
Challenges

1. No entry point
2. Kernel core model
3. No stable API
4. Low level code
   Pointer Analysis with Uninterpreted Functions
   M. Mandrykin
5. Hardware specific
6. Concurrency
Challenges

(1) No entry point

(2) Kernel core model
Using Aspect-Oriented Programming for Preparing C Programs for Static Verification
E. Novikov

(3) No stable API
Using Aspect-Oriented Programming for Preparing C Programs for Static Verification
E. Novikov

(4) Low level code
Pointer Analysis with Uninterpreted Functions
M. Mandrykin

(5) Hardware specific

(6) Concurrency
Challenges

(1) No entry point
*Environment model generator*

(2) Kernel core model
*Using Aspect-Oriented Programming for Preparing C Programs for Static Verification*
E. Novikov

(3) No stable API
*Using Aspect-Oriented Programming for Preparing C Programs for Static Verification*
E. Novikov

(4) Low level code
*Pointer Analysis with Uninterpreted Functions*
M. Mandrykin

(5) Hardware specific

(6) Concurrency
Pseudo-main generation

```c
int main(int argc, char* argv[]) {
    init_module();
    for(;;) {
        switch(*) {
            case 0: driver_probe(*,*,*);break;
            case 1: driver_open(*,*);break;
            ...
        }
    }
}
exit_module();
```
Challenges

(1) No entry point
Environnent model generator

(2) Kernel core model
Using Aspect-Oriented Programming for Preparing C Programs for Static Verification
E. Novikov

(3) No stable API
Using Aspect-Oriented Programming for Preparing C Programs for Static Verification
E. Novikov

(4) Low level code
Pointer Analysis with Uninterpreted Functions
M. Mandrykin

(5) Hardware specific

(6) Concurrency
Future Data Race Detection Tools

- Runtime analysis
  - Kernel Strider (Google Research Award)
    - KEDR(ISPRAS) + ThreadSanitizer(Google)
  - Race Hound
    - HW breakpoints
    Beta to be released by the end of 2012 for x86 only

- Static analysis
  - Research in progress
Conclusions

(1) No entry point

Environment model generator

(2) Kernel core model

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification
E. Novikov

(3) No stable API

Using Aspect-Oriented Programming for Preparing C Programs for Static Verification
E. Novikov

(4) Low level code

Pointer Analysis with Uninterpreted Functions
M. Mandrykin

(5) Hardware specific

No idea

(6) Concurrency

To be researched
Still positive

Crete, 2005
Thank you!

Alexey Khoroshilov
khoroshilov@linuxtesting.org
http://linuxtesting.org/project/ldv