```
BUG: unable to handle kernel NULL pointer dereference at virtual address 0000009c
 printing eip:
c01e41ee
*pde = 00000000
Oops: 0000 [#1]
SMP
Modules linked in:
CPU:    0
EIP:    0060:[<c01e41ee>]    Not tainted VLI
EFLAGS: 00010202   (2.6.18-1-k7 #1)
EIP is at acpi_hw_low_level_read+0x7/0x6a
eax: 00000010   ebx: 00000001   ecx: 00000094   edx: c18e1f80
esi: c18e1f94   edi: 00000000   ebp: 00000000   esp: c18e1f68
ds: 007b   es: 007b   ss: 0068
Process swapper (pid: 1, ti=c18e0000 task=f7b44aa0 task.ti=c18e0000)
Stack: 00000001 c18e1f94 00000000 c01e42ae 00fb3c00 00000000 00000000 c02b670c
       f7fb3c00 c02b6834 c01c21b5 c02b66dc c01c1e26 f7fb3c00 c0344b6c 00000000
       c01c12d0 00000000 c01003e1 c0102b46 00000202 c01002d0 00000000 00000000
Call Trace:
 [<c01e42ae>] acpi_hw_register_read+0x5d/0x177
 [<c01c21b5>] quirk_via_abnormal_poweroff+0x11/0x36
 [<c01c1e26>] pci_fixup_device+0x68/0x73
 [<c01c12d0>] pci_init+0x11/0x28
 [<c01003e1>] init+0x111/0x28e
 [<c0102b46>] ret_from_fork+0x6/0x1c
 [<c01002d0>] init+0x0/0x28e
 [<c01002d0>] init+0x0/0x28e
 [<c0101005>] kernel_thread_helper+0x5/0xb
Code: a0 82 2d c0 76 1b 50 68 85 8c 2a c0 68 f3 00 00 00 ff 35 ac ef 28
c0 e8 c7 80 00 00 31 d2 83 c4 10 89 d0 c3 57 85 c9 56 53 74 5d <8b>
71 08 8b 59 04 89 f7 09 df 74 51 c7 02 00 00 00 00 8a 09 84
EIP: [<c01e41ee>] acpi_hw_low_level_read+0x7/0x6a SS:ESP 0068:c18e1f68
 <0>Kernel panic - not syncing: Attempted to kill init!
```

Motivation

- My last PLASMA talk: Case study on using BLAST in order to find and avoid common programming errors in Linux device drivers (Mühlberg and Lüttgen, 2006)
- Results:
  - Substantial amount of time for manual modification of the source code is required
  - No support for pointer operations
  - No support for concurrency

Simulation-based Verification of Memory Safety Properties for Object Code Programs

Jan Tobias Mühlberg
muehlber@cs.york.ac.uk

University of York, UK

PLASMA Seminar, York, 22nd February 2007

Simulation-based Verification of Memory Safety Properties for **Object Code Programs**

Why Object Code?

- Programs are not always available in source code (proprietary stuff, libraries)
- Do properties hold after compilation and optimisation?
- Many bugs exist because of platform specific details
- Programs may be modified after compilation
- Unspecified language constructs, use of inline assembly or multiple languages

Simulation-based Verification of **Memory Safety Properties** for Object Code Programs

Memory Safety?

- Supported by some verification suites:
  - `NULL`-pointer dereferences
  - alternating calls of *malloc()* and *free()*

- Problem:
  - What if a pointer does not equal `NULL` but is *invalid*?
  - What if we are operating on nested data structures?

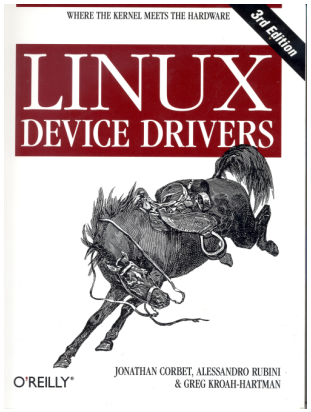
Memory Safety? (cont'd)

"[...] we define a software entity to be *memory safe* if (a) it never references a memory location outside the the address space allocated by or for that entity, and (b) it never executes instructions outside the code area created by the compiler and linker within that address space." (Dhurjati et al., 2005)

Memory Safety? (cont'd)

- What I am interested in:
  - Dereferencing invalid pointers
  - Uninitialised reads
  - Buffer overflows
  - Memory leaks
  - Violation of API usage rules for (de)allocation functions

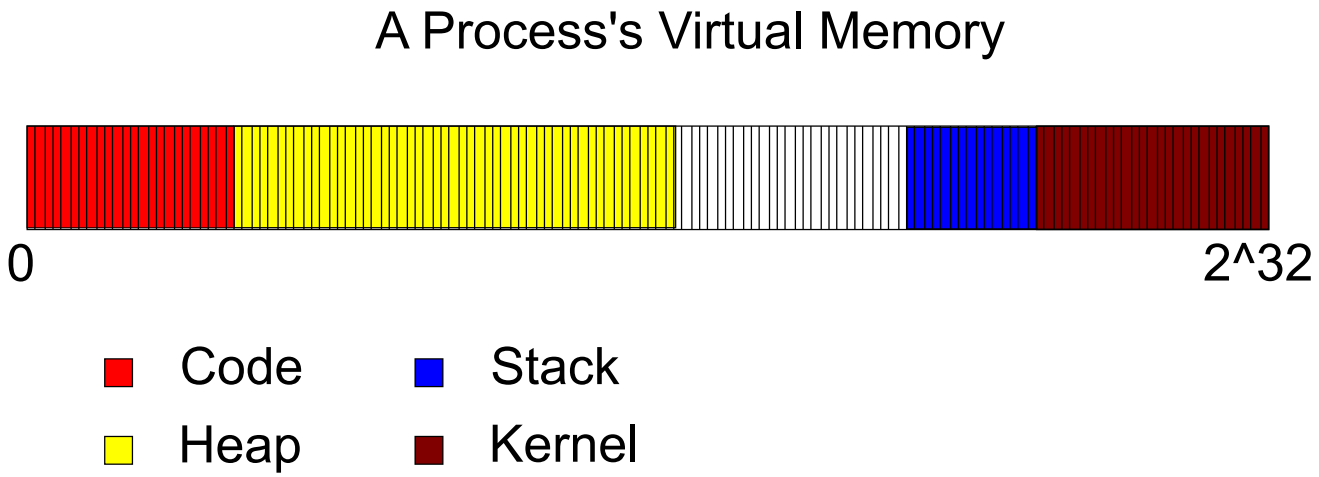
Memory Safety? (cont'd)



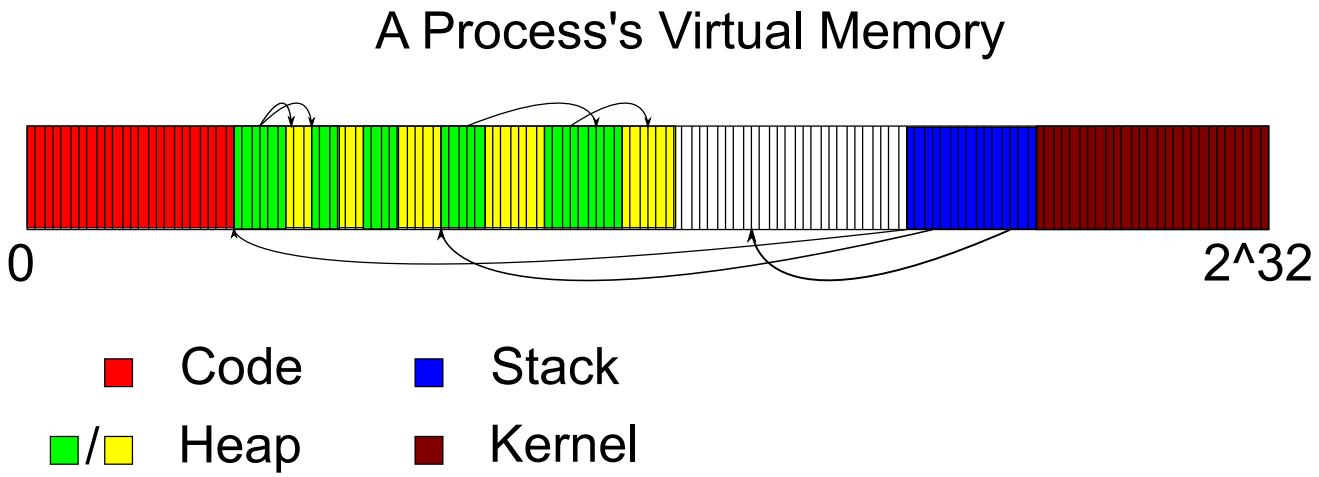
"You should never pass anything to *kfree* that was not obtained from *kmalloc*." (Corbet et al., 2005)

- Several APIs for allocation and de-allocation
- General problems: Is a pointer valid at any time it is de-referenced?
  Is it valid for a specific operation?

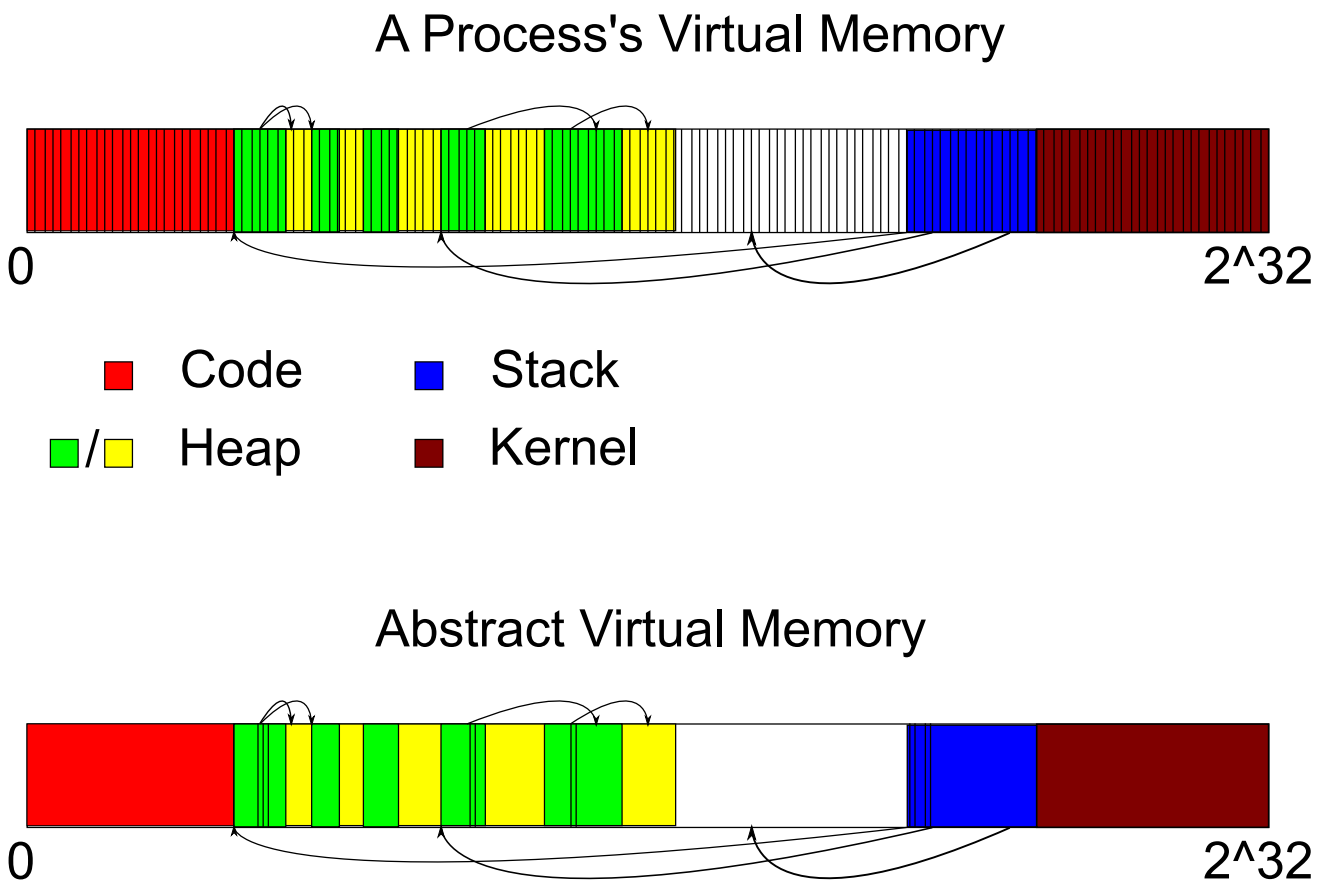
Memory Safety? (cont'd)



Memory Safety? (cont'd)



Memory Safety? (cont'd)



Memory Safety? (cont'd)

- What I am interested in:
  - Dereferencing invalid pointers
  - Uninitialised reads
  - Buffer overflows
  - Memory leaks
  - Violation of API usage rules for (de)allocation functions

Simulation-based Verification of Memory Safety Properties for Object Code Programs

Simulation-based Verification?

- We have a complex software system and we are interested in properties that can probably not be verified directly
- However, we can test the system or *simulate* its behaviour

Simulation-based Verification?

- Can we prove that our test bench or the set of simulation runs covers all important situations?
- Interesting approaches for using coverage metrics in combination with constraint solving and model checking (Chockler et al., 2001), (Chockler et al., 2003)

Why Linux Device Drivers?

- Highly critical domain
- Modular software architecture
- Small programs with high complexity
- Almost no tool support for debugging and verification
- Plenty of case studies available to compare results with

My Approach

1. Generate a small runtime environment for the driver
   - Identify module dependencies
   - Provide implementations for all symbols referenced in the modules under consideration
   - Generate functions that call the driver's interface functions with valid parameters
   - Link a static binary

My Approach

1. Generate a small runtime environment for the driver
2. Analyse the resulting binary
   - Identify code, data, entry points, basic blocks
   - That is undecidable – we have to be conservative

My Approach

1. Generate a small runtime environment for the driver
2. Analyse the resulting binary
3. Generate IR and DFG
   - Explicit load/store operations
   - Static single assignment form
   - Based upon Valgrind's VEX library

My Approach

1. Generate a small runtime environment for the driver
2. Analyse the resulting binary
3. Generate IR and DFG
4. Program slicing
   - We are not interested in most computations
   - We are not interested in I/O operations
   - Eliminates $\sim$ 60% of a driver's instructions

My Approach

1. Generate a small runtime environment for the driver
2. Analyse the resulting binary
3. Generate IR and DFG
4. Program slicing
5. Generate and solve constraint system
   - Use DFG after slicing as a dependency graph
   - Not yet implemented!

My Approach

1. Generate a small runtime environment for the driver
2. Analyse the resulting binary
3. Generate IR and DFG
4. Program slicing
5. Generate and solve constraint system
6. Generate a set of initial memory states
   - Currently: memory areas are filled with random data

My Approach

1. Generate a small runtime environment for the driver
2. Analyse the resulting binary
3. Generate IR and DFG
4. Program slicing
5. Generate and solve constraint system
6. Generate a set of initial memory states
7. Run the resulting programs

Examples. . .

- I would like to present something,

  but that's another talk...

Future Work

- Formalise data-flow analysis and slicing techniques used; give more talks
- Implement proper generation of initial memory states using constraint solving
- Add support for concurrent executions of a driver's code
- Employ the tool in a case study on file systems (with Andy and Gerald)

Future Work

- Make the whole thing publicly available
- Use model checking to verify the exhaustiveness of the simulation?
- Can we apply further abstraction and partitioning to use model checking directly?

Thank you!

References

Chockler, H., Kupferman, O., Kurshan, R., and Vardi, M.: 2001, A practical approach to coverage in model checking, in *Computer Aided Verification, Proc. 13th International Conference*, Vol. 2102 of *Lecture Notes in Computer Science*, pp 66–78, Springer-Verlag

Chockler, H., Kupferman, O., and Vardi, M.: 2003, Coverage metrics for formal verification, in *12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, Vol. 2860 of *Lecture Notes in Computer Science*, pp 111–125, Springer-Verlag

Corbet, J., Rubini, A., and Kroah-Hartmann, G.: 2005, *Linux Device Drivers*, O'Reilly, Sebastopol, CA, USA, 3rd edition

Dhurjati, D., Kowshik, S., Adve, V., and Lattner, C.: 2005, *Trans. on Embedded Computing Sys.* **4(1)**, 73

Mühlberg, J. T. and Lüttgen, G.: 2006, Blasting linux code, in *FMICS 2006*, No. 4346 in LNCS, pp 211 – 226

Random other things

- Izura mailed: "Please send my warmest regards to everybody." And she updated her publications list. . .
- Publications list: Izura, Gerald, Malcolm, Mike, Tobias and Alan updated their lists recently. How about the others?
- The term is over in three weeks. We need talks for next term!
- Next talk. . .