1,000,000 (LOC) and Counting

Static Analysis for Errors and Vulnerabilities in the Linux Kernel Source Code

Peter T. Breuer & Simon Pickin
Universidad Carlos III de Madrid

Piet Hein - Grooks

"A needle in a haystack
may be difficult to find;
your chance of ever
finding one is small.
Especially with haystacks
of the ordinary kind,
which don't have any
needles in at all."

Goal

ë Apply

Mathematical Methods

to the source code of the

Linux kernel

Must be

ë post hoc

ë *capable of application by nonexperts*

ë *able to handle* 6.5 *million lines of* rapidly changing *C*

Sleep under Spinlock Hunt (SluSH)

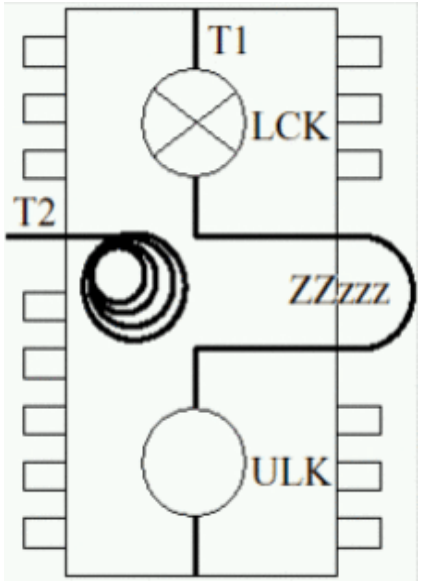
files checked:	1055	
alarms raised:	18	(5/1055 files)
false positives:	16/18	
real errors:	2/18	(2/1055 files)
time taken:	~24h	
LOC:	~700K	(unexpanded)

What is "sleep under spinlock"?

Sleep - thread scheduled out of CPU

Spinlock - busy wait for lock

- "2+2 = 1"
  - 2 CPUs + 2 threads busy waiting
    = 1 dead machine



Output from SluSH run

```
total 1 instances of sleep under spinlock in
                    sound/isa/sb/sb16_csp.c
total 1 instances of sleep under spinlock in
                    sound/oss/sequencer.c
total 6 instances of sleep under spinlock in
                    net/bluetooth/rfcomm/tty.c
total 7 instances of sleep under spinlock in
                    net/irda/irlmp.c
total 3 instances of sleep under spinlock in
                    net/irda/irttp.c
```

Output summarises liklihoods

```
#include <linux/autoconf.h>     (2425 lines)
#include <linux/config.h>       (8 lines)
...
#include <linux/blk.h>          (407 lines)
#include "ioctls.h"             (15 lines)
#include "sbull.h"              (67 lines)
#include <linux/blkpg.h>        (66 lines)

      function      line    symbol          (severity)

 <linux/smb_fs_sb.h>
  smb_lock_server   66      down                (0)

 <linux/fs.h>
  lock_parent       1620    down                (0)
  double_down       1646    down                (0)
  triple_down       1684    down                (0)
  double_lock       1712    double_down         (0)

 <linux/locks.h>
  lock_super        40      down                (0)

 "sbull.c"
  sbull_ioctl       184     interruptible_sl..  (0)
* sbull_request     420     interruptible_sl..  (1)
  sbull_init        469     kmalloc             (0)


* found 1 instances of sleep under spinlock
```

Example of bad code

snd_sb_csp_load() in sb16_csp.c

```
 ...
spin_lock_irqsave(&p->chip->reg_lock, flags);
 ...
unsigned char *kbuf, *_kbuf;
_kbuf = kbuf = kmalloc (size, GFP_KERNEL);
 ...
```

Another piece of guilty code

Kernel 2.6.12 sound/oss/sequencer.c
midi_outc()

```
spin_lock_irqsave(&lock,flags);
while (n && !midi_devs[dev]->outputc(dev, data)) {
    interruptible_sleep_on_timeout(&seq_sleeper, HZ/25);
    n--;
}
spin_unlock_irqrestore(&lock,flags);
```

Cox owns up

```
Return-Path: <alan@lxorguk.ukuu.org.uk>
Subject: Re: sleep under spinlock, sequencer.c, 2.6.12.5
Cc: linux kernel <linux-kernel@vger.kernel.org>
Date: Fri, 19 Aug 2005 19:07:09 +0100
[...]
Yep thats a blind substitution of lock_kernel in an old
tree it seems.  Probably my fault. Should drop it before
the sleep and take it straight after.
```

Other problem classes ...

Access (read/write) to kfreed memory

Overflow 4096B of stack

Spinlock under spinlock

Call to function that expects non NULL parameters with possibly NULL argument

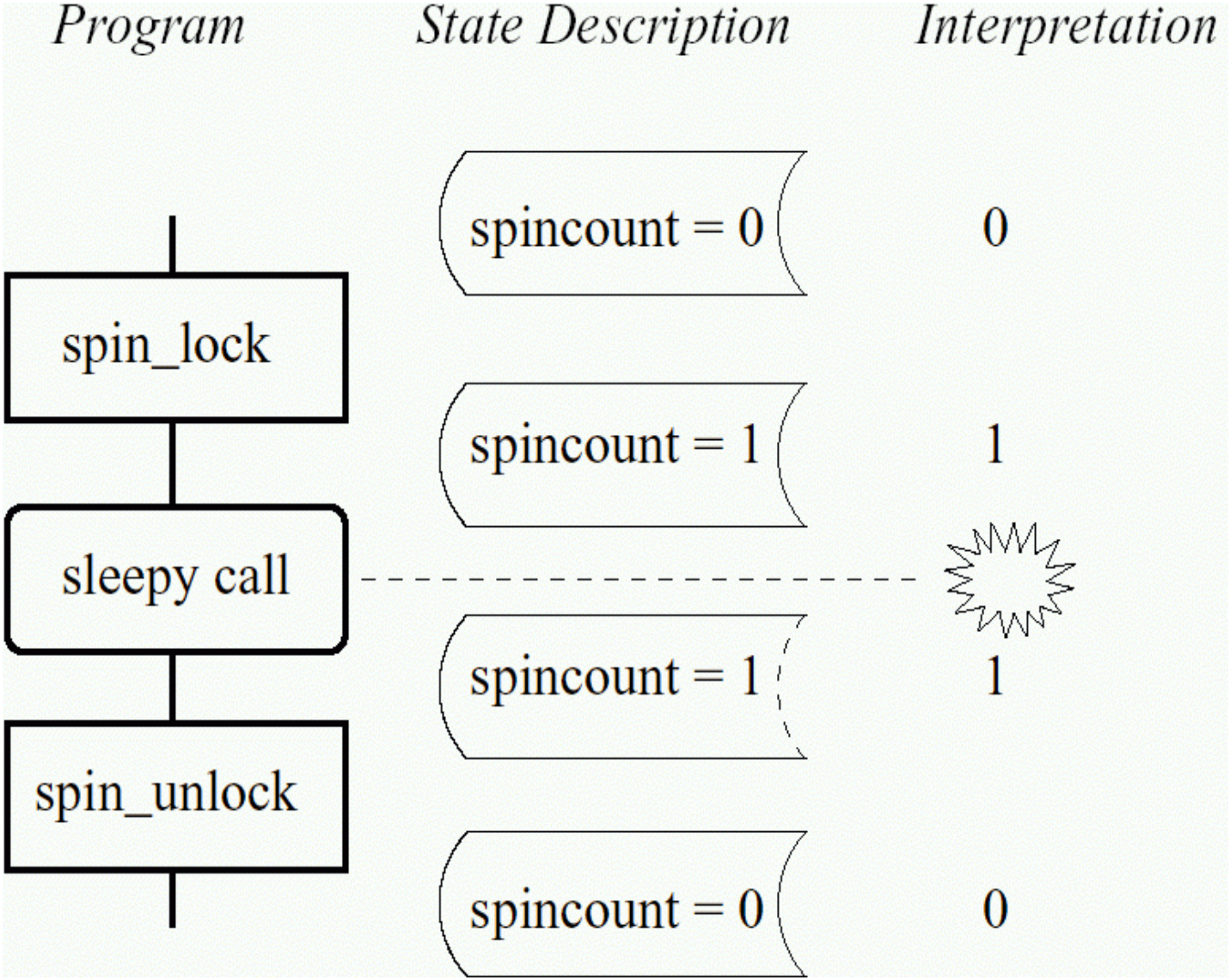
Logic is *configured*, so new tests can be invented

Example of kfree/access

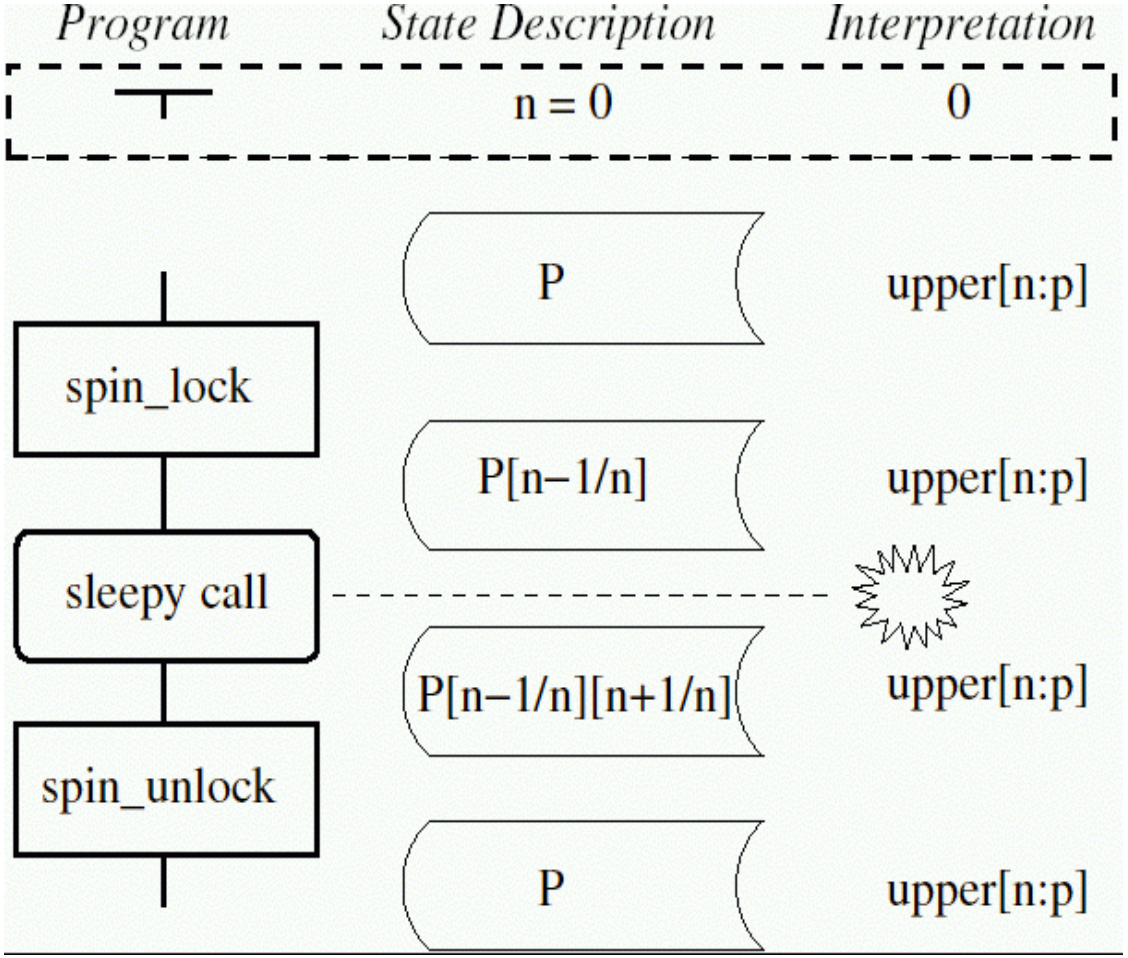
drivers/scsi/aix7xxx_old.c in kernel 2.6.3

```
while(current_p && temp_p) {
  if (((current_p->pci_bus==temp_p->pci_bus)&&...
    ...
    kfree(temp_p);
    continue;
    ...
  }
  ...
}
```

Basic technique



The abstract view



Symbolic Approximation

Description of statements as logic transformers

p **x.count=x.count+1** p[n-1/n]

p ◊^ **spin_trylock(&x)** p[n-1/n] ◊ 1 | p ◊ 0

Approximation of programs

More approximate program, weaker logic for reasoning about it

More specified, can say more about program

Choice of approximation is choice of logic

Other aspects of system

Symbolic approximation provides theory

- *class of abstract interpretations with R*

Different perspectives of each approximation

- Trigger/action system for raising alarms!

Compositional logic NRBG

normal, return, break, goto

- Adjusting logic adjusts approximation

NRB - Statement Logic

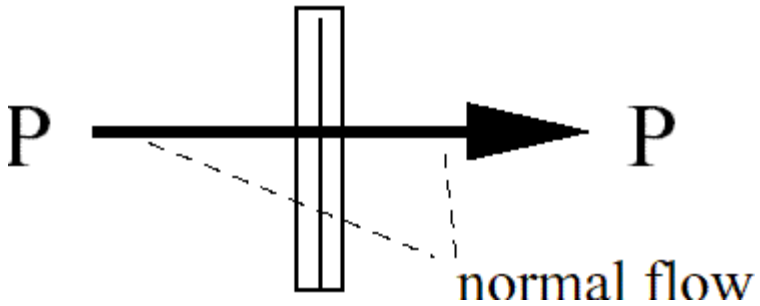
Empty statement

```
ctx e, p::empty() = (p, F, F) with ctx e;
```

maintains condition p *normally* (p)

empty statement cannot return (F)

empty statement cannot break (F)

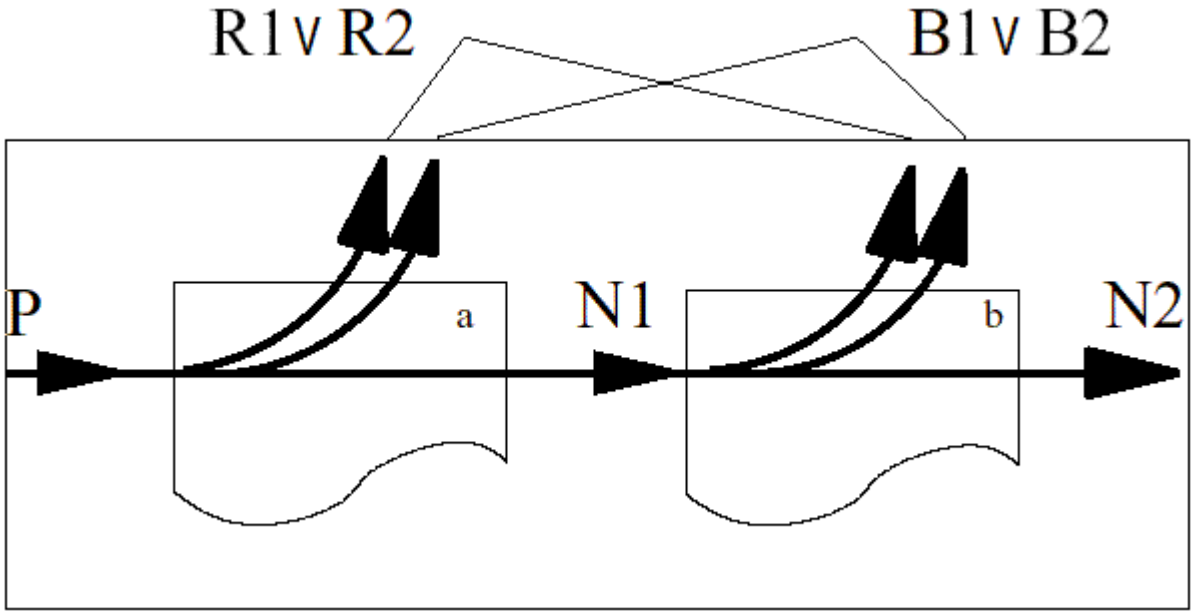


Sequence logic -NRB

normal exit: traverse A then B

return exit: return from A
OR traverse A then return from B

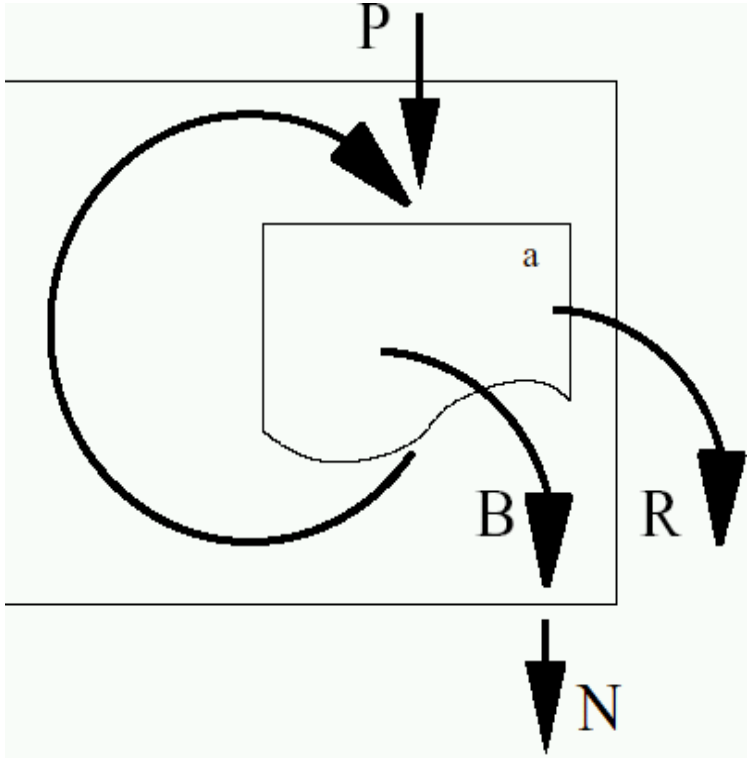
break exit: break from A
OR traverse A then break from B



NRB - Forever Loop logic

break from body is *normal* exit from **while(1)**

relax* p until it is *invariant




```
ctx e, p::while1(a) = (b, r, F) with ctx f
  where ctx e, p:: a = (n, r, b) with ctx f;
```

Programmable trigger/action engine

Three rules handle propagation of call graph and other housekeeping.

a sleep call while the objective function is positive causes output:

$$(\texttt{SLEEP} \;\&\; \texttt{OBJECTIVE\_SET} \;\&\; \texttt{OBJECTIVE} \geq 0)! \rightarrow \texttt{output}()$$

Using the analyser

Call with the same arguments as given to the gcc compiler

```
c -Wp,-MD,arch/i386/kernel/.msr.o.d -nostdinc \
  -iwithprefix include -D__KERNEL__ -Iinclude \
  -D__KERNEL__ -Iinclude -Wall -Wstrict-prototypes \
  -Wno-trigraph s -fno-strict-aliasing -fno-common \
  -pipe -mpreferred-stack-boundary=2 -march=i686 \
  -Iinclude/asm-i386/mach-default -O2 \
  -fomit-frame-pointer -DMODULE -DKBUILD_BASENAME=msr \
  -DKBUILD_MODNAME=msr -c -o arch/i386/kernel/msr.o \
   arch/i386/kernel/msr.c
```

Limitations

- Predicates are restricted to unions of n-cubes
  - checking if p ® q NP-complete problem
- State is not followed well enough:
  - x = 1; if (x) A else B;
    - treated correctly - only A is evaluated
  - if (x) A else B; if (x) C else D;
    - generally over-abstracted - A;C | A;D | B;C | B;D
  - solution is to push branch hypotheses down ((x≠0);A | (x=0);B) ; ((x≠0);C | (x=0);D)
    - hypotheses not always calculable

Example of symbolic approximation

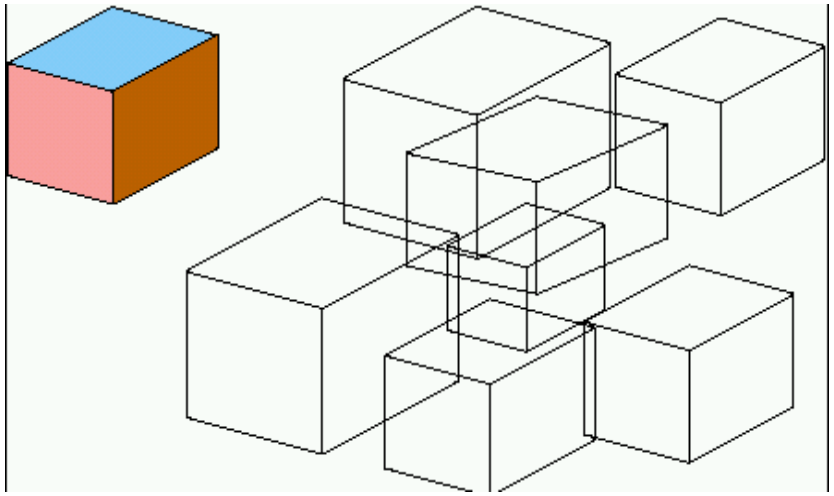
$$\frac{p \rhd t \quad \mathtt{x} \quad ((x \neq 0) \wedge q_1 \rhd x \;[]\; (x = 0) \wedge q_0 \rhd x) \qquad (x \neq 0) \wedge q_1 \rhd x \quad \mathtt{a;} \quad r_1 \rhd t_1 \qquad (x = 0) \wedge q_0 \rhd x \quad \mathtt{b;} \quad r_0 \rhd t_0}{p \rhd t \quad \mathtt{if\ (x)\ a;\ else\ b;} \quad r_1 \vee r_0 \rhd \top}$$

- Branch hypotheses q_0 q_1
  - weak logic
    - $q_0 \leftrightarrow q_1 \leftrightarrow$ True
  - strong logic (exact)
    - e.g. $q_0 \leftrightarrow \exists u,v.\ u^2 + v^2 = w^2$
      $q_1 \leftrightarrow \forall u,v.\ u^2 + v^2 \neq w^2$

Implication of predicates is decidable

Basic evaluation is $C \Rightarrow \bigcup C_i$ of cubes

i.e. $\bigcup C_i$ *covers* C



Summary

- A step towards analyses of 100MLOC.
  - No expertise needed
  - Fast
  - Safe
  - Copes with massive amounts of code
- Negatives
  - (deliberately) not perfect tracking program state
    - *symbolic approximation* provides the theory/context
  - Needs expert to extend to new problem classes