Linux Kernel Auditing

Alan Cox

Red Hat

Why Do We Want To Audit Code ?

- All Programmers Make Mistakes
  - Programmers rarely see their own errors
  - A second viewpoint may see better methods
- Good Practice Changes Over Time
  - Hardware assumptions change over time
  - Old code needs updating to newer APIs
- Security Auditing
  - In some ways a side effect of code auditing

Alan's First Rule Of Open Source

“Criticism Is Infinitely Scalable”

C Language Traps

- Maths overflow does not cause an exception
  - Be very careful to verify the range before multiplying or adding to user provided values
  - Remember that signed overflow is undefined
- Strings require that space for termination
- Structures can be padded by the compiler
  - Use attribute(“packed”) with caution
- Endianness is your problem

Object Lifespan

- For any allocated object can you clearly define
  - Where it is created
  - Where it is destroyed
  - The point at which you know all use of it is finished
- Does the object have a common creation function
- Are all fields initialized at creation
  - Slab poisoning
- Are all resources freed on all paths

Portability

- Use u8/u16/u32 for fixed type sizes
- Check endianness in the code
- Use unsigned long for memory and I/O resources
  - Otherwise Dave Miller gets annoyed
- Check that ioremap memory is
  - Used with readb/writeb and friends
  - Not dereferenced directly
  - Not used with memcpy/memset

Use of DMA

- Beware of old code using virt_to_bus with DMA
- DMA is not allowed to target
  - Vmalloc memory (including module data)
  - High pages
  - The stack
- When fixing DMA abuse use the new API
- Check DMA always finishes before freeing
  - DMA scribbles into free memory are horrible

The PCI Bus

- Update drivers to the PCI hotplug API when possible
  - Much more important for 2.5
- PCI is message based
  - Interrupts and PCI data transfer are asynchronous
  - PCI writes may be delayed
  - PCI to main memory ordering is not totally obvious
- PCI devices may have 64bit addressing

Resources Are Finite

- System Calls
  - Allocates user controlled amounts of memory
  - Queues undefined numbers of buffers of user data
- Interrupt Handling
  - Make sure CPU time is bounded
  - Beware infinite memory allocations
- The Stack
  - Typically you have 7K, sometimes 4K for IRQ, 4K for system call context.

Deadlocks

- Spin locks are not recursive
- Sleeping with a spinlock held is not permitted
  - 2.5 has debugging traps for this
  - Copying to/from user space sleeps
  - Do not replace long sleeps with mdelay() waits
- Beware of IRQ/system call deadlocks
  - You cannot disable_irq holding spinlocks taken by an IRQ handler

Termination Order

- Ensure DMA is finished before freeing memory
- Use del_timer_sync to ensure timers are done
  - Being careful about deadlocks
- Do not free PCI interrupts until the IRQ is block on the card
  - Allowing for PCI posting
- Terminate any created threads
  - Ensure they are using complete_and_exit

Documentation

- Auditing code requires understanding it
  - Document anything that was not documented
  - Switch existing documentation to kernel-doc
  - Make it clear which release any audit was done for
- If you discover a new common error
  - Report it to the kernel list
  - Help other people fix the other instances

Submitting Changes

- Seperate functional and stylistic change
  - Submit things like reindenting to coding style first
- Try and submit each functional change as a seperate patch or changeset
  - Makes debugging easier
  - Makes understanding changes easier
  - Makes rejecting one change of a set easier
  - Gets code accepted