Diagnosys : Automatic Generation of a Debugging Kernel Interface for Linux Services

Work in progress

Tegawendé F. Bissyandé, Laurent Réveillère (LaBRI)

Julia Lawall, Gilles Muller (INRIA/LIP6)

The problem: Developing Linux kernel services

- There are bugs in kernel services
  - Drivers + File systems + sound
    - 400-550 faults per release [ASPLOS 2011]
- State of the art
  - Formal specifications (seL4)
    - Not for the "ordinary" engineer
  - Static analysis
    - Long-running tools
    - (Too) many reports to check
- Need solutions for the standard Linux developer

Refining the problem

- Not all developers are expert in the core kernel
  - Interactions between the kernel and the developed service
  - Wrong parameters/return values may crash/hang the kernel or the developed service
    - Safety holes
- The kernel evolves continuously and so the safety holes
  - No well-defined list of exported function

Our approach

- Identify the list of safety holes for a given Linux version
  - Based on exported symbols
  - Usage preconditions
- Track safety hole violations by a service
  - Provide precise debugging messages to the developers
  - Make the message log crash persistent across reboot

Tools

- Safety Hole Analyzer : SHAna
  - Kernel static analysis
  - Categorization based on [SOSP2001] and [ASPLOS2011]
  - Run once per kernel version by a dedicated maintainer
- Debugging Interface Generator : DIGen
  - Identification of functions used by the service
  - Generation of an interface
  - Easing the compilation with the debugging interface
- Diagnosys logging system : Dlog
  - Crash resilient logging system coupled with Kexec fast reboot system
  - User-space utility for reserving memory and accessing logs

"Entry" Safety Hole

- Service code

```
tx_skb = dev_alloc_skb(pkt_len);
+ if (!tx_skb) { . . . goto out; }
pkt_data = skb_put(tx_skb, pkt_len);
```

- Kernel code

```
unsigned char *skb_put(struct sk_buff *skb, . . .)
{. . . skb->tail += len; . . .}
```

"Exit" Safety Hole

- Kernel code

```
struct block_device *open_bdev_exclusive(. . .)
{. . . return ERR_PTR(error);}
```

- Service code

```
bdev = open_bdev_exclusive(. . .);
- if (!bdev) return -EIO;
+ if (IS ERR(bdev)) return PTR_ERR(bdev);
...
filemap_write_and_wait(bdev->bd_inode->i_mapping);
```

Other examples

- Implicit free (exit safety hole)

```
- netif_rx(skb);
lp->stats.rx bytes += skb->len; . . .
+ netif_rx(skb);
```

- Bad lock usage (entry safety hole)

```
- spin_lock(&inode->i lock);
res = nfs_scan_commit(. . .);
- spin_unlock(&inode->i lock);
```

Safety Holes in Linux 2.6.32

- SHAna reported 22,940 safety holes in 7,505 exported functions

Safety hole	Number of exported functions	
	Entry	Exit
Block	367	815
IsNull/Null	7 220	1 124
Var	5	11
Lock/Intr/LockIntr	815	23
Free	-	11
Size	8	-
Range	-	8

Potential impact of the categorized safety holes

Total number of commits in Linux 2.6	278 078
Commits related to exported functions	11 294
Commits related to the usage of exported function	703
Commits related to the categorized safety holes	287

- 38% of commits (267/703) are related to the categorized safety holes

Benefit of Logs

- Linux Backtrace

[847.353202] BUG: unable to handle kernel paging request at ffffffee
[847.353205] IP: [<fbc722d9>] btrfs init new device+0xcf/0x5c5 [btrfs]
[847.353229] *pdpt = 00000000007ee001 *pde = 00000000007ff067
[847.353233] Oops: 0000 [#1] . . .
[847.353291] EIP is at btrfs init new device+0xcf/0x5c5 [btrfs] . . .
[847.353298] Process btrfs-vol (pid: 3699, . . .
[847.353312] Call Trace:
[847.353327] [<fbc7b84e>] ? btrfs ioctl add dev+0x33/0x74 [btrfs]
[847.353334] [<c01c52a8>] ? memdup user+0x38/0x70 . . .
[847.353451] ---[end trace 69edaf4b4d3762ce]---

- Diagnosys

[jiffies] File.c:line | 'open_bdev_exclusive' returned an invalid pointer | EXIT | ERR_PTR

Coverage of the runtime checks

- Missing NULL and ERR_PTR tests after memory allocation

Category	Kernel module	# of mutations	# of crashes with			Coverage
			No log	Log is not last	Log is last	
Networking	e1000e	57	0	0	20	100%
	iwlagn	18	1	0	8	88.9%
	btusb	9	1	0	7	87.5%
USB	Usb-storage	11	0	0	3	100%
	Ftdi_sio	9	0	0	6	100%
Multimedia drivers	Snd-intel8x0	3	1	0	2	66.7%
	uvcvideo	34	3	3	17	73.9%
File systems	Isofs	28	3	0	9	75%
	Nfs	309	13	9	157	87.7%
	fuse	77	3	1	41	91.1%

Conclusion

- An approach for easing the understanding of a kernel misuse by a service developer
- Approach integrates seamlessly with current developer habits and expertise
- Ongoing work on increasing the log information and faults detected