Diagnosys: Automatic Generation of a Debugging Interface to the Linux Kernel

Tegawendé F. Bissyandé, Laurent Réveillère
(University of Bordeaux)
Julia Lawall, Gilles Muller (INRIA/LIP6-Regal)

June 27, 2012

Bugs: They're everywhere!

Bugs in software

Null pointer dereference:

```
if (!std || ...) {
  if (std->len % qset->max_packet != 0)
    return -EINVAL;
  ...
}
```

Bugs in software

Null pointer dereference:

```
  if (!std || ...) {
    if (std->len % qset->max_packet != 0)
      return -EINVAL;
    ...
  }
```

Use after free:

```
  if (radio->disconnected) {
    ...
    kfree(radio);
    goto unlock;
  }
  ...
unlock:
  mutex_unlock(&radio->disconnect_lock);
```

Bugs: What to do about them?

Static bug-finding tools

- Coccinelle, Coverity, FindBugs, SDV, Astree, etc.

Dynamic bug-finding tools

- Valgrind, KLEE, testing, etc.

Bugs: What to do about them?

Static bug-finding tools

- Coccinelle, Coverity, FindBugs, SDV, Astree, etc.

Dynamic bug-finding tools

- Valgrind, KLEE, testing, etc.

These tools require complete programs, containing all code fragments related to the bug.

Problem: Bugs arising across the plugin/core boundary

Plugin:

```
tx_skb = dev_alloc_skb(pkt_len);
pkt_data = skb_put(tx_skb, pkt_len);
```

Core:

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

- Plugin code is buggy:
  - Should check for NULL.
- Core code is not robust:
  - Dereferences its argument without checking.
- Bug in plugin leads to crash in core:
  - Core is not well-known to plugin developers.
  - Core contains a safety hole.

Issues

Should core exported functions be more robust?

- Plugging safety holes comes at a performance cost.

Should the core export a specific robust interface?

- Requires a stable set of exported functions.
- Induces maintenance.
- Limits evolvability.

The right choice is application dependent.

- We focus on Linux, which
  - Does not export a fixed interface
  - Does not require exported functions to be robust.

Further Linux issues

- Core code is large and complex.
- Many exported functions, most of which are undocumented.
- Debugging mostly relies on backtraces.
  - Unreliable.
  - Limited context information.

```
[  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 ]---
```

Our solution: Diagnosys

Goal: Rather than expect the Linux core developers to construct and maintain a debugging interface, generate one automatically.

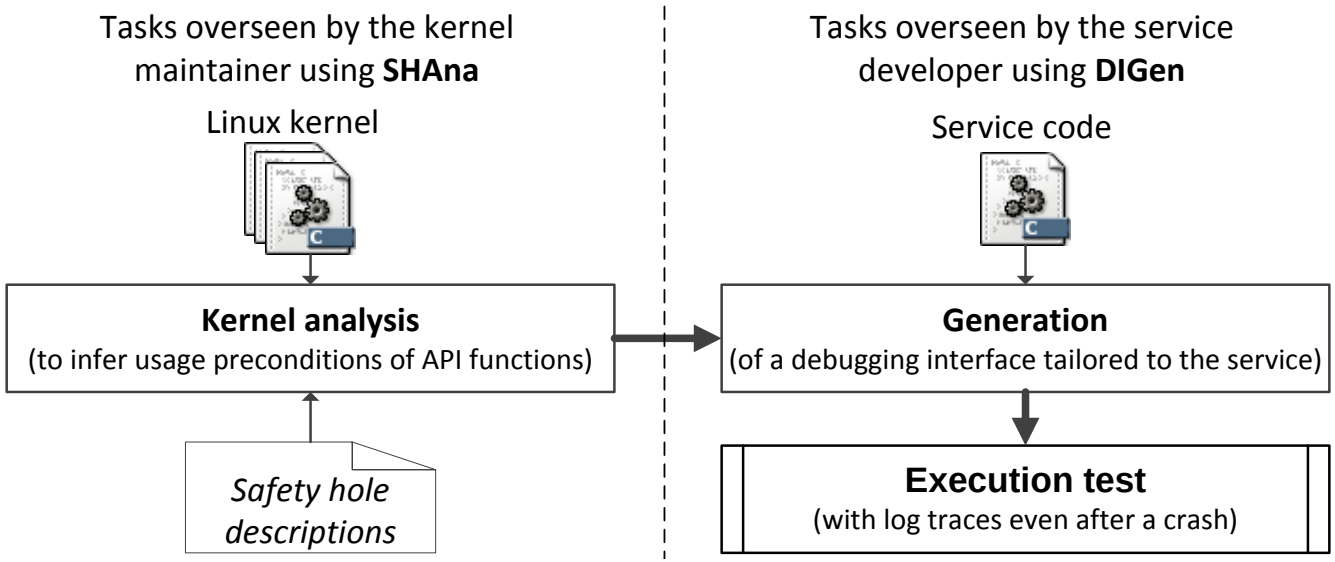
Automatically produce a readable log of dangerous operations along the plugin/core boundary that may lead to a crash or hang.

Our solution: Diagnosys

Goal: Rather than expect the Linux core developers to construct and maintain a debugging interface, generate one automatically.

Automatically produce a readable log of dangerous operations along the plugin/core boundary that may lead to a crash or hang.

Diagnosys architecture:



SHAna: Identifying kernel exported functions

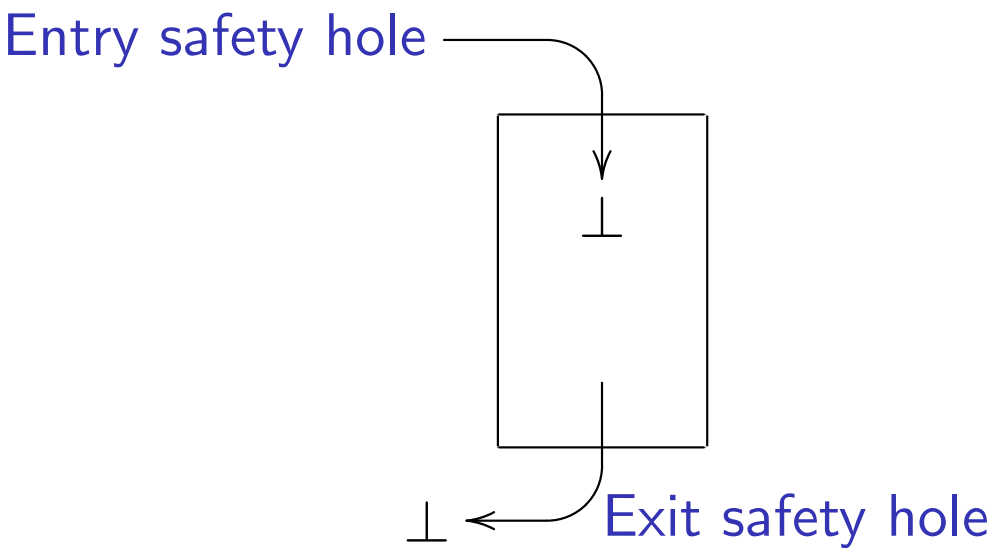
Kernel exported functions are declared as:

- `EXPORT_SYMBOL(f)`
- `EXPORT_SYMBOL_GPL(f)`

SHAna: Identifying safety holes

Safety hole: Code fragment within a core function that introduces the possibility of a bug across the core/plugin boundary.

- Entry safety holes are certain or possible.
- Exit safety holes are always possible.

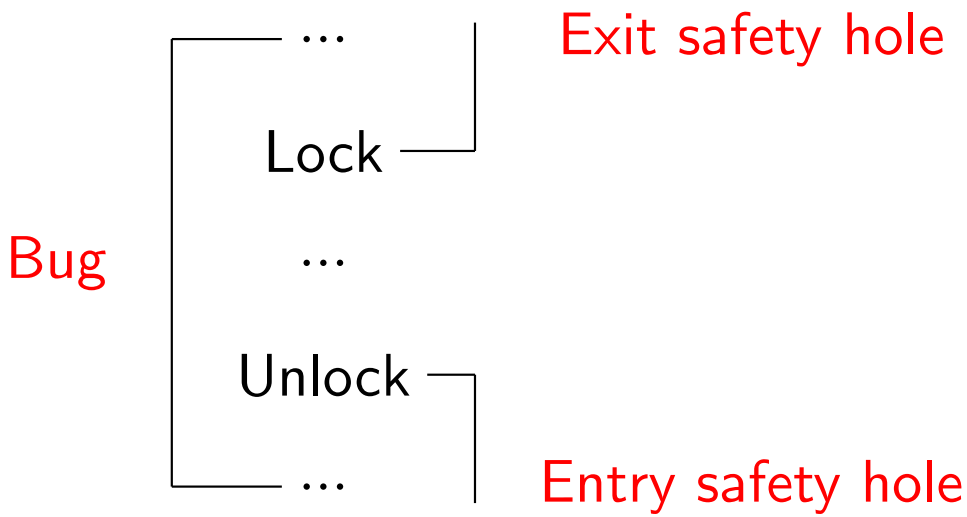


SHAna: Identifying safety holes

Observation: Any bug type that involves multiple disjoint code fragments can lead to an entry or exit safety hole.

- We consider bug types identified by Chou et al. [SOSP01]
- Interprocedural analysis.

Example:



DIGen: The generated code

```
static inline <rtype> __debug_<kernel function> (...) {
  <rtype> __ret;
  /* Check preconditions for entry safety holes */
  if <an entry safety-hole safety precondition is violated>
     diagnosys_log(<EF id>, <SH cat>, <info (e.g., arg number)>);
  /* Invocation of the intended kernel function */
  __ret = <call to kernel function>;
  /* Check preconditions for exit safety holes */
  if <an exit safety-hole safety precondition is violated>
     diagnosys_log(<EF id>, <SH cat>, <info (e.g., err ret type)>);
  /* Forward the return value */
  return __ret;
}

#define <kernel function> __debug_<kernel function>
```

The developer's view

1. SHAna is run once by a kernel maintainer.
   - Results published for download.
2. Developer compiles plugin code using a dedicated make script (`dmake`).
   - Identifies kernel exported functions.
   - Invokes DIGen.
3. DIGen generates a header file containing a robust wrapper for each used kernel exported function.
4. `dmake` recompiles the code, including the header file.

Steps 2-4 are transparent once the developer invokes `dmake`.

Evaluation

- Scope of the problem
- Improvement in debuggability
  - Qualitatively
  - Quantitatively
- Performance overhead

Scope of the problem

Linux 2.6.32	Number of exported functions collected in the	
Safety hole	entry sub-category	exit sub-category
Block	367	815
INull/Null	7,220	1,124
Var	5	11
Lock/Intr/LockIntr	815	23
Free	-	11
Size	8	-
Range	-	8

Safety holes identified using Coccinelle.

- Around 400 false positives, mostly due to multiple architecture-specific function definitions.

About half of the in-kernel calls to kernel exported functions call functions with safety holes.

Qualitative improvement in debuggability

A bug in btrfs code (and its fix)

```
  bdev = open_bdev_exclusive(...);
- if (!bdev) return -EIO;
+ if (IS_ERR(bdev)) return PTR_ERR(bdev);
```

Kernel gives a backtrace from wherever `bdev` is dereferenced.

Diagnosys reports on previous dangerous operations.

```
...
[4294934950]|@/var/diagnosys/tests/my_btrfs/volumes.c:1441|
                    open_bdev_exclusive|INULL(EXITED)|ERR PTR|
```

Quantitative improvement in debuggability

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 drivers	usb-storage	11	0	0	3	100%
	ftdi_sio	9	0	0	6	100%
Multimedia device drivers	snd-intel8x0	3	1	0	2	66.7%
	uvcvideo	34	3	3	17	73.9%
File systems	isofs	28	3	0	9	75.0%
	nfs	309	13	9	157	87.7%
	fuse	77	3	1	41	91.1%

- Mutations remove NULL/IS_ERR tests.
- Inject allocation failures when initializing the tested value.

Quantitative improvement in debuggability

In 230 oops reports derived from mutation tests (NULL and lock)

- Diagnosys log contains information about the mutation 92% of the time.
- Debugging without Diagnosys required consulting 1 to 14 functions, in up to 4 files.
- Diagnosys often halves this number.

Performance overhead

Network driver: netperf benchmark

Test		**Without Diagnosys**	**With Diagnosys**	**Overhead**
TCP_STREAM	Throughput	907.91 Mb/s	904.32 Mb/s	0.39%
UDP_STREAM	Throughput	951.00 Mb/s	947.73 Mb/s	0.34%
UDP_RR	Throughput	7371.69 Tx/s	6902.81 Tx/s	6.36%

File system: IOzone benchmark

Record block size(Kb)	**Without Diagnosys** (Access rate - K/sec) read/write	**With Diagnosys** (Access rate - K/sec) read/write	**Overhead** read/write
128	45309/31672	42141/28072	6.99%/11.36%
256	49780/36577	48196/32900	3.18%/10.05%
512	49764/39957	45765/37981	8.03%/4.94%

Conclusion

- Developing plugins for a large code base is a challenge for developers.
  - Documentation not up to date.
  - Crashes/hangs hard to interpret.
- We have identified safety holes as a probable source of difficulties.
- We propose Diagnosys to automatically generate wrappers that log dangerous uses of functions that contain safety holes.
- Usable in practice:
  - On mutation tests, reduces the amount of work to find bugs.
  - Low performance overhead (no impact on in-kernel calls).