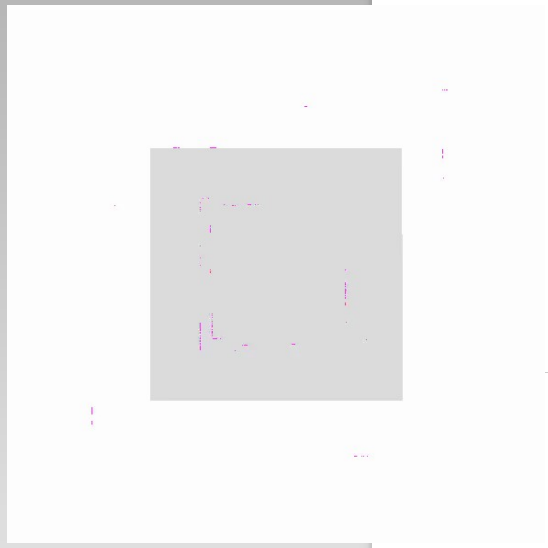
technische universität
dortmund

Improving Linux-Kernel Tests for LockDoc with Feedback-driven Fuzzing

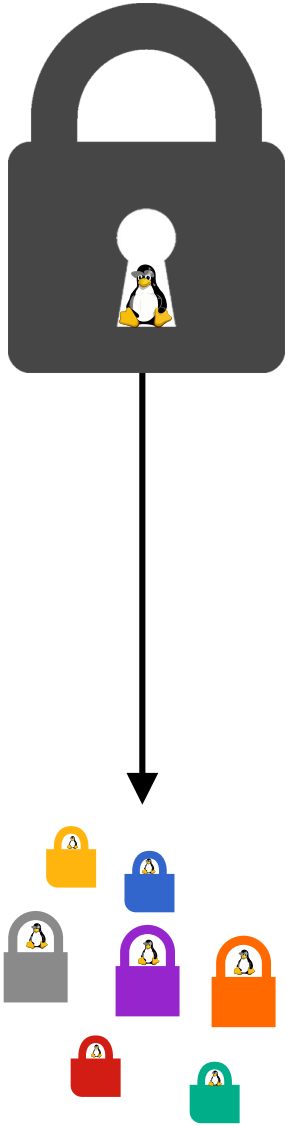


Alexander Lochmann, Robin Thunig,
Horst Schirmeier

alexander.lochmann@tu-dortmund.de
https://ess.cs.tu-dortmund.de/~al

Embedded System Software Group
Computer Science 12, TU Dortmund

Locking the Linux Kernel



- Shift from **Big Kernel Lock** to a **fine-grained locking scheme** [1]
  - **Reduce** lock **contention**
  - **Better scaling** on multi-core platforms
  - Needs **good documentation** for developers
- Locking Documentation …
  - is usually scattered throughout the source code
  - does not use a well-defined syntax
  - is ambiguous or even conflicting

→ Inefficiencies, deadlocks, crashes, silent data corruptions

What is LockDoc?

- Tracks **locking pattern** and **data-structure** accesses
- Validates/generates documentation, and locates locking bugs
- **Recording** performed under a **load**
  - Relies on how system is put under load
  - **Wants** to reach even **remote parts** of the code

- LockDoc study [2]
  - Focuses on *Virtual File System* (VFS) subsystem
  - Uses filesystem-specific subset of *Linux Test Project* (LTP)

Alexander Lochmann, Horst Schirmeier, Hendrik Borghorst, and Olaf Spinczyk. 2019. *LockDoc: Trace-Based Analysis of Locking in the Linux Kernel*. EuroSys'19.

What is LockDoc?

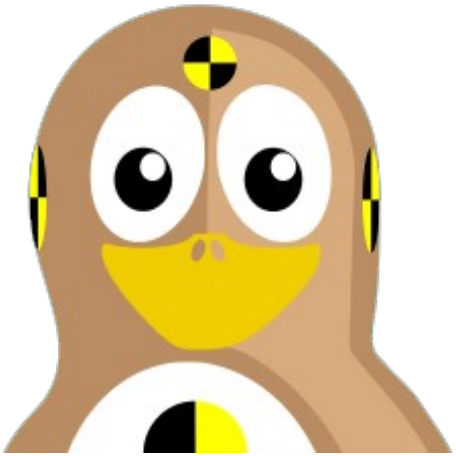
- Tracks **locking pattern** and **data-structure** accesses
- Validates/generates documentation, and locates locking bugs
- **Recording** performed under a **load**
  - Relies on how system is put under load
  - **Wants** to reach even **remote parts** of the code
- LockDoc study [2]
  - Focuses on *Virtual File System* (VFS) subsystem
  - Uses filesystem-specific subset of *Linux Test Project* (LTP)

→ How much source code is covered?

Alexander Lochmann, Horst Schirmeier, Hendrik Borghorst, and Olaf Spinczyk. 2019. *LockDoc: Trace-Based Analysis of Locking in the Linux Kernel*. EuroSys'19.

What is the L(inux) T(est) P(roject)?

- LTP “validate(s) the reliability, robustness, and stability of Linux” [3]
- **Framework** for Linux **Kernel Tests** via the **System Call Interface**
- Collection of **manually crafted regression tests** for various subsystems, e.g., IPC, VFS, scheduling, ...
- Individual tests are composed to test suites, e.g., *syscalls*



Code Coverage by LTP – A Breakdown

- Run test suites related to VFS
- Measure basic block coverage
- Linux 4.10
  - Total BBs: 342,732
  - VFS BBs: 75,531
- Only **~35%** of kernel's basic blocks are executed by **all VFS-related testsuites** of LTP

Test Suite	#Tests	Covered VFS BBs	(%)
dio	30	8312	11.0%
fcntl-locktests	1	2420	3.2%
filecaps	1	2518	3.3%
fs	65	17495	23.2%
fs_ext4	4	13081	17.3%
fs_perms_simple	18	5081	6.7%
fsx	1	6572	8.7%
io	2	6817	9.0%
syscalls	1181	24217	32.1%
Total	1302	26229	34.7%

Code Coverage by LTP – A Breakdown

- Run test suites related to VFS
- Measure basic block coverage
- Linux 4.10
  - Total BBs: 342,732
  - VFS BBs: 75,531
- Only **~35%** of kernel's basic blocks are executed by **all VFS-related testsuites** of LTP

Test Suite	#Tests	Covered VFS BBs	(%)
dio	30	8312	11.0%
fcntl-locktests	1	2420	3.2%
filecaps	1	2518	3.3%
fs	65	17495	23.2%
fs_ext4	4	13081	17.3%
fs_perms_simple	18	5081	6.7%
fsx	1	6572	8.7%
io	2	6817	9.0%
syscalls	1181	24217	32.1%
Total	1302	26229	34.7%

Code Coverage by LTP – A Breakdown

- Run test suites related to VFS
- Measure basic block coverage
- Linux 4.10
  - Total BBs: 342,732
  - VFS BBs: 75,531
- Only **~35%** of kernel's basic blocks are executed by **all VFS-related testsuites** of LTP

Test Suite	#Tests	Covered VFS BBs	(%)
dio	30	8312	11.0%
fcntl-locktests	1	2420	3.2%
filecaps	1	2518	3.3%
fs	65	17495	23.2%
fs_ext4	4	13081	17.3%
fs_perms_simple	18	5081	6.7%
fsx	1	6572	8.7%
io	2	6817	9.0%
syscalls	1181	24217	32.1%
Total	1302	26229	34.7%

Code Coverage by LTP – A Breakdown

- Run test suites related to VFS
- Measure basic block coverage
- Linux 4.10
  - Total BBs: 342,732
  - VFS BBs: 75,531
- Only **~35%** of kernel's basic blocks are executed by **all VFS-related testsuites** of LTP

Test Suite	#Tests	Covered VFS BBs	(%)
dio	30	8312	11.0%
fcntl-locktests	1	2420	3.2%
filecaps	1	2518	3.3%
fs	65	17495	23.2%
fs_ext4	4	13081	17.3%
fs_perms_simple	18	5081	6.7%
fsx	1	6572	8.7%
io	2	6817	9.0%
syscalls	1181	24217	32.1%
Total	1302	26229	34.7%

Code Coverage by LTP – A Breakdown

- Run test suites related to VFS
- Measure basic block coverage
- Linux 4.
  - Tota
  - VFS BBs: 75,531
- Only **~35%** of kernel's basic blocks are executed by **all VFS-related testsuites** of LTP

→ How can we cover more code?

Test Suite	#Tests	Covered VFS BBs	(%)
dio	30	8312	11.0%
fcntl-locktests	1	2420	3.2%
		2518	3.3%
		17495	23.2%
fs_ext4	4	13081	17.3%
fs_perms_simple	18	5081	6.7%
fsx	1	6572	8.7%
io	2	6817	9.0%
syscalls	1181	24217	32.1%
Total	1302	26229	34.7%

What is syzkaller?

- Coverage-guided kernel fuzzer [4]
- Wants to **trigger kernel bugs**
- Fuzzes Linux kernel by **randomly generating user programs**:

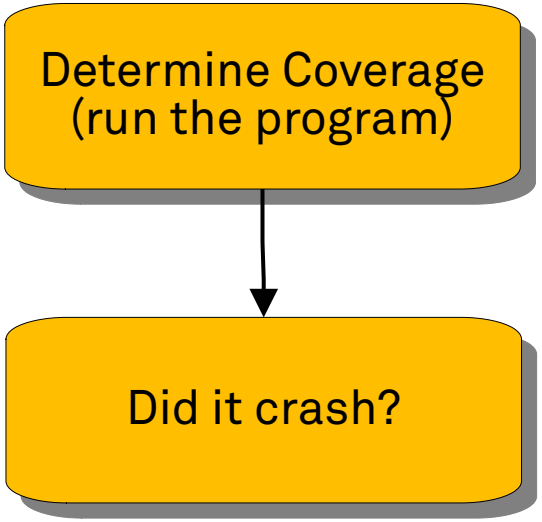
```
int main(void)
{
  syscall(__NR_mmap, 0x1ffff000, 0x1000, 0, 0x32, -1, 0);
  syscall(__NR_mmap, 0x20000000, 0x1000000, 7, 0x32, -1, 0);
  syscall(__NR_mmap, 0x21000000, 0x1000, 0, 0x32, -1, 0);

  *(uint32_t*)0x20002480 = 0x20000340;
  memcpy((void*)0x20000340, "\x12", 1);
  *(uint32_t*)0x20002484 = 1;
  *(uint32_t*)0x20002488 = 0;
  syz_read_part_table(0, 1, 0x20002480);
  return 0;
}
```

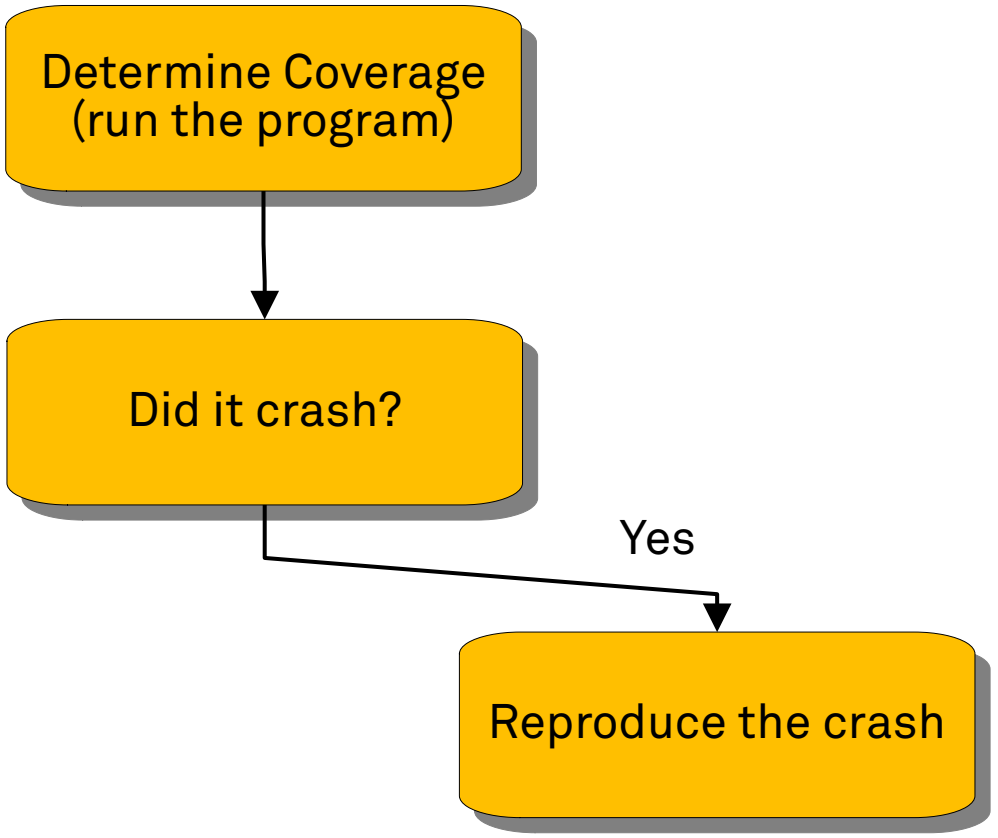
How does syzkaller work?

Determine Coverage
(run the program)

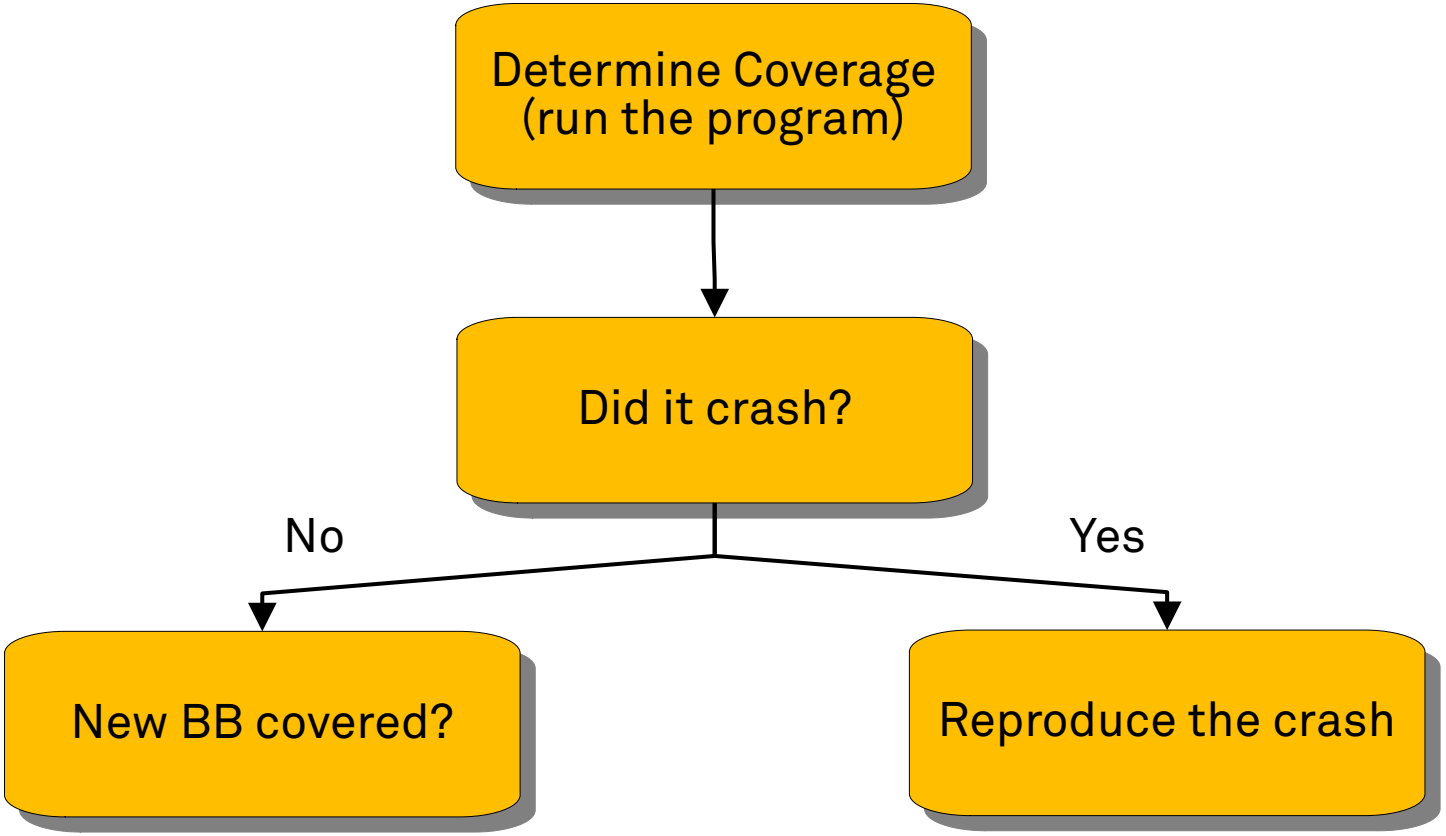
How does syzkaller work?



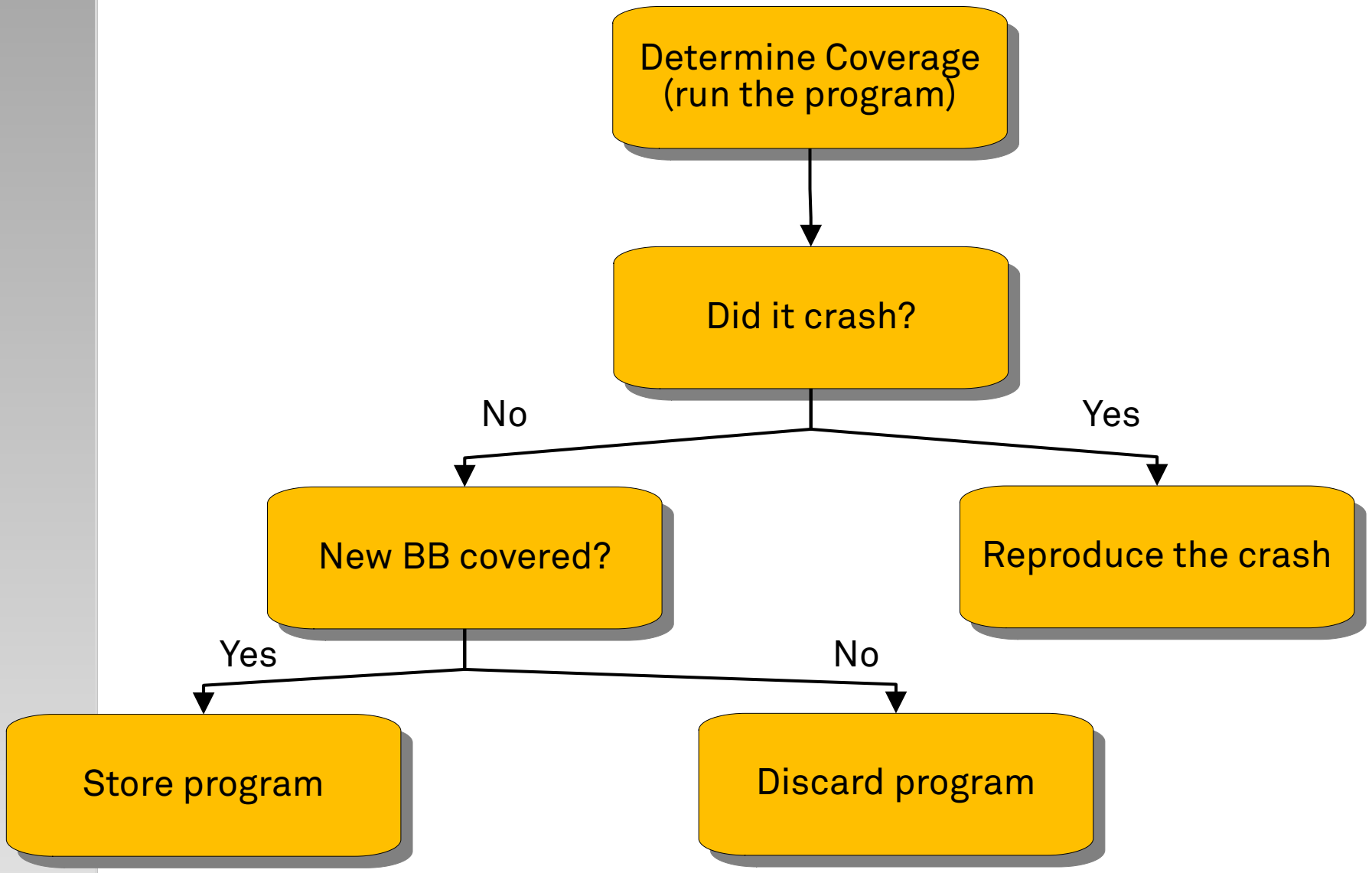
How does syzkaller work?



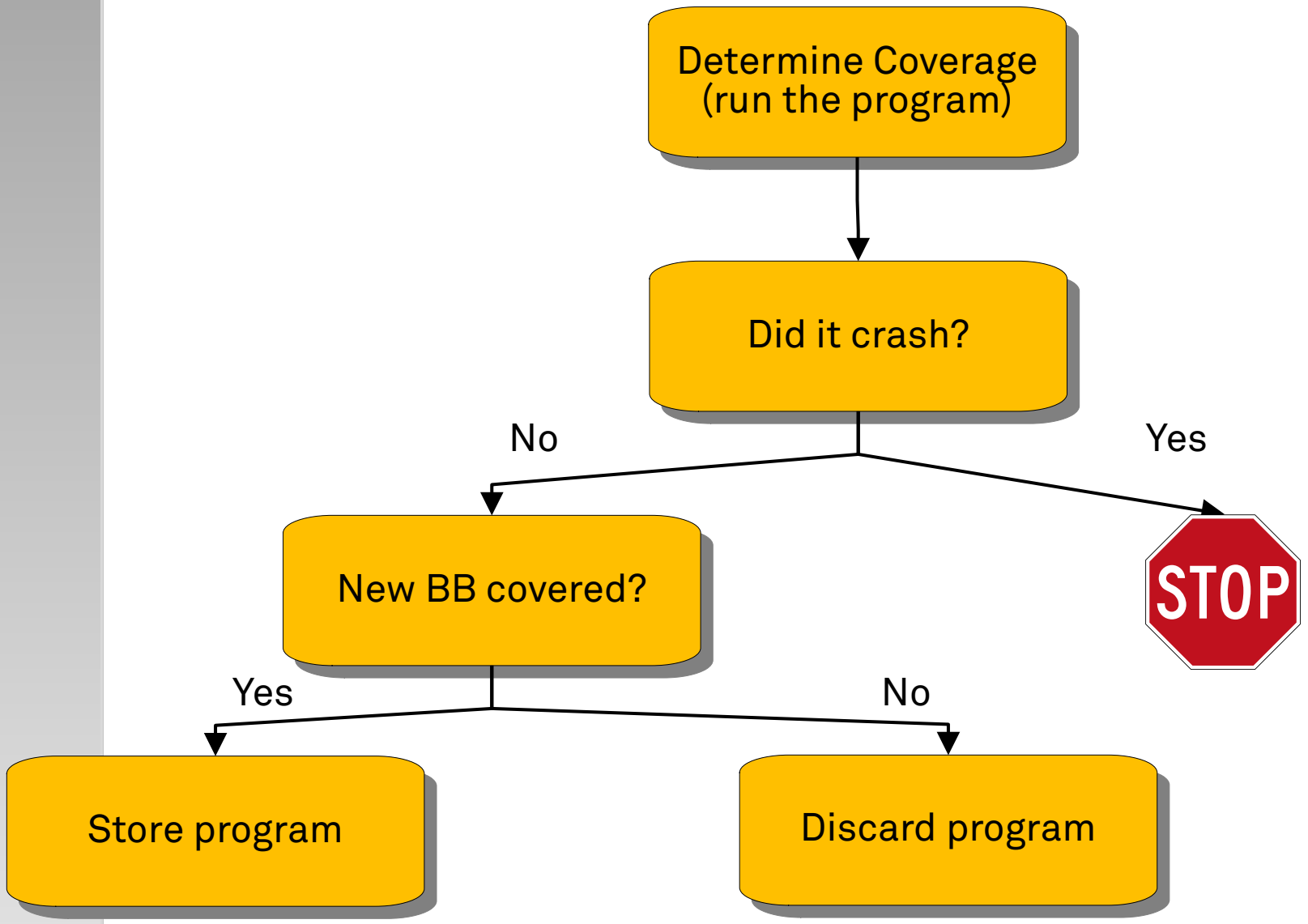
How does syzkaller work?



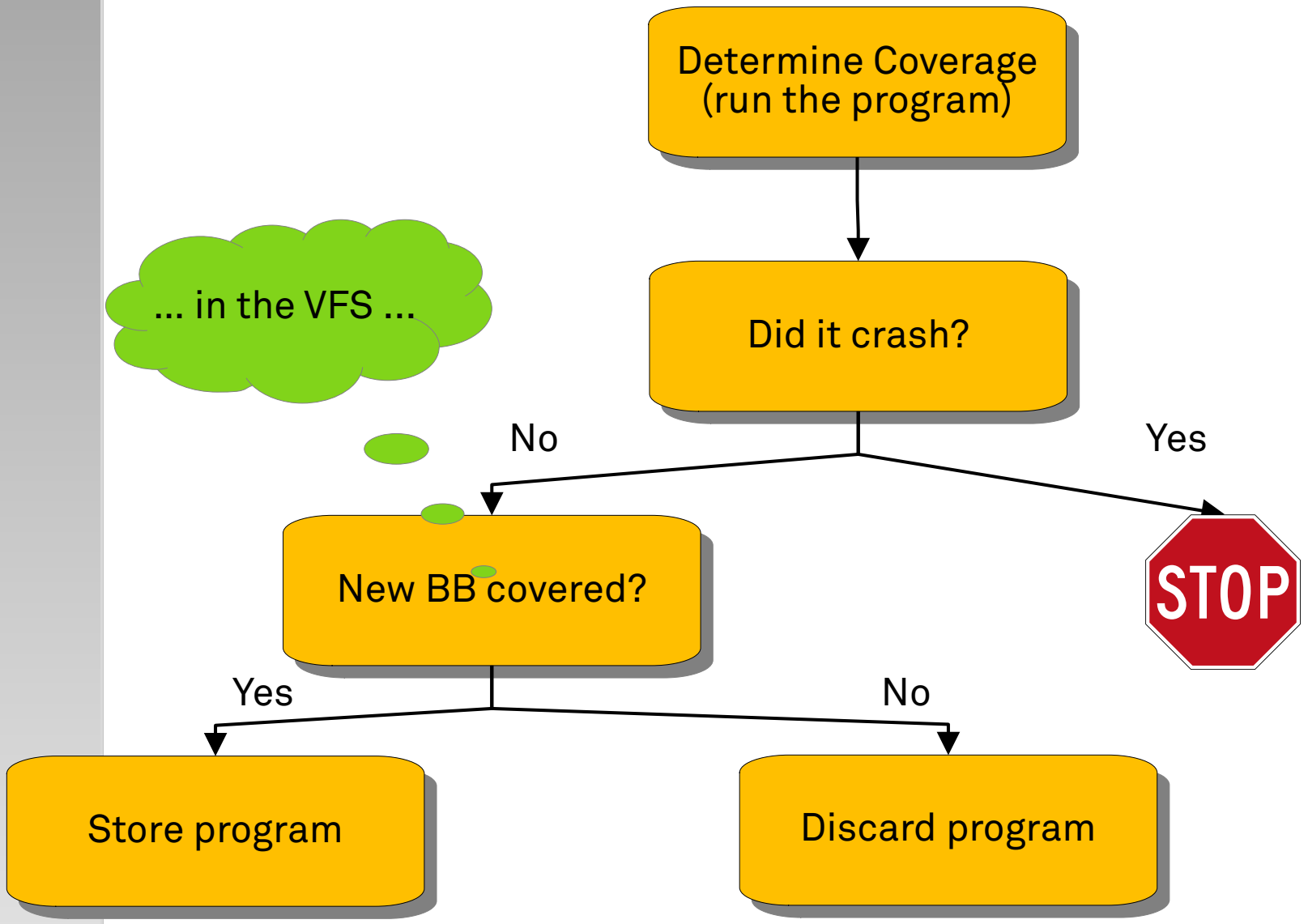
How does syzkaller work?



How does syzkaller work?



How does syzkaller work?

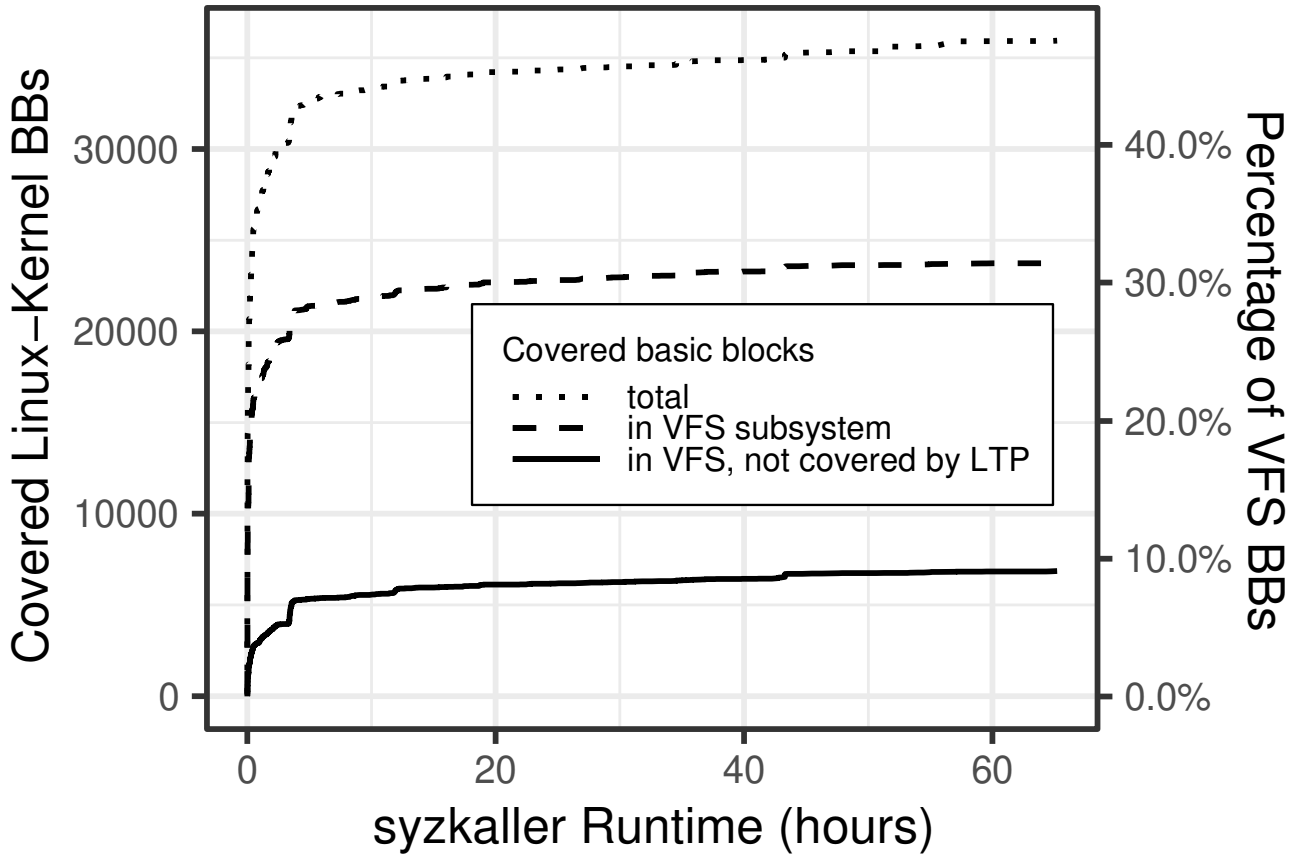


Evaluation Setup

- Linux Kernel 4.10 x86 64-bit
  - Without module support
  - Minimal configuration: Network support, essential drivers for root fs and for a paravirtualized environment
- Record executed basic blocks via Vyukov's KCOV [5]
- Use of LD_PRELOAD to collect covered BBs for a process hierarchy
- Convert BB to source code via *addr2line*
- Filtering using regex: */fs/|/mm/|fs\.h|mm\.h*

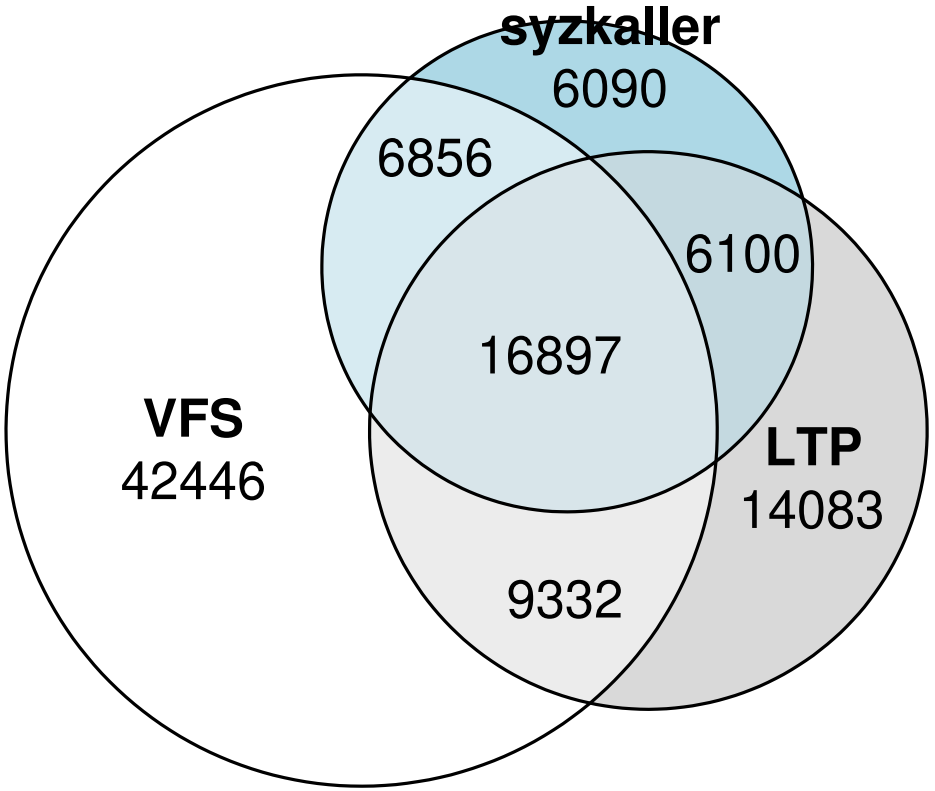
syzkaller Activity

Results (1)



- 65-hour run: syzkaller generated 2278 programs
- BB coverage: 10% of whole kernel, 31.4% of VFS
- 9.1% of VFS BBs covered that are not covered by LTP

Results (2)



- Relation between BBs: VFS vs. syzkaller vs. LTP
- Absolute numbers of basic blocks shared among intersecting sets

Summary

- Basic-block coverage of VFS by LTP is limited
- Used modified syzkaller to reach more VFS-related code
- Increased basic-block coverage by 9.1 % to 43.8%
- More observations (mem. accesses/lock ops.) for LockDoc to analyze → Increased quality
- Outlook
  - Use *Moonshine* [6] approach to improve code coverage
  - Generate full-fledged regression tests, sent them back to LTP?

References

[1] Robert Love. 2010. *Linux Kernel Development (3rd ed.)*.

[2]Alexander Lochmann, Horst Schirmeier, Hendrik Borghorst, and Olaf Spinczyk. 2019. *LockDoc: Trace-Based Analysis of Locking in the Linux Kernel*. EuroSys‘19.

[3] https://github.com/linux-test-project/ltp

[4] https://github.com/google/syzkaller

[5]https://www.kernel.org/doc/html/latest/dev-tools/kcov.html

[6]Shankara Pailoor, Andrew Aday, and Suman Jana. 2018. *MoonShine: Optimizing OS Fuzzer Seed Selection with Trace Distillation*. USENIX Security Symposium 2018.