Application Crash Vulnerabilities

Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnatthan Alagappan,
Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

File-System Crash Consistency

- Ensures **logical consistency of internal metadata**
- Important for FS developers and researchers
- Much research, multiple techniques
  - FSCK, Soft Updates, Journaling, COW …

Application-Level Crash Consistency

- Applications don't use internal metadata of FS
- What happens to **user data during a system crash?**
- Maintain application-defined consistency on user data structures: Application-Level Consistency

State of the art

For effective application-level consistency, applications depend on **specific details** of file-system implementation

- Bad situation
  - Many file systems in use
  - New file systems constantly invented
- Application-level consistency is important
  - Modern applications store many data structures
  - Google Chrome initialization: 500+ files
    - Data structures like page cache, history
    - Cache should have only complete entries
  - Photo application: Thumbnails match pictures

Example: Atomic File Rewrite

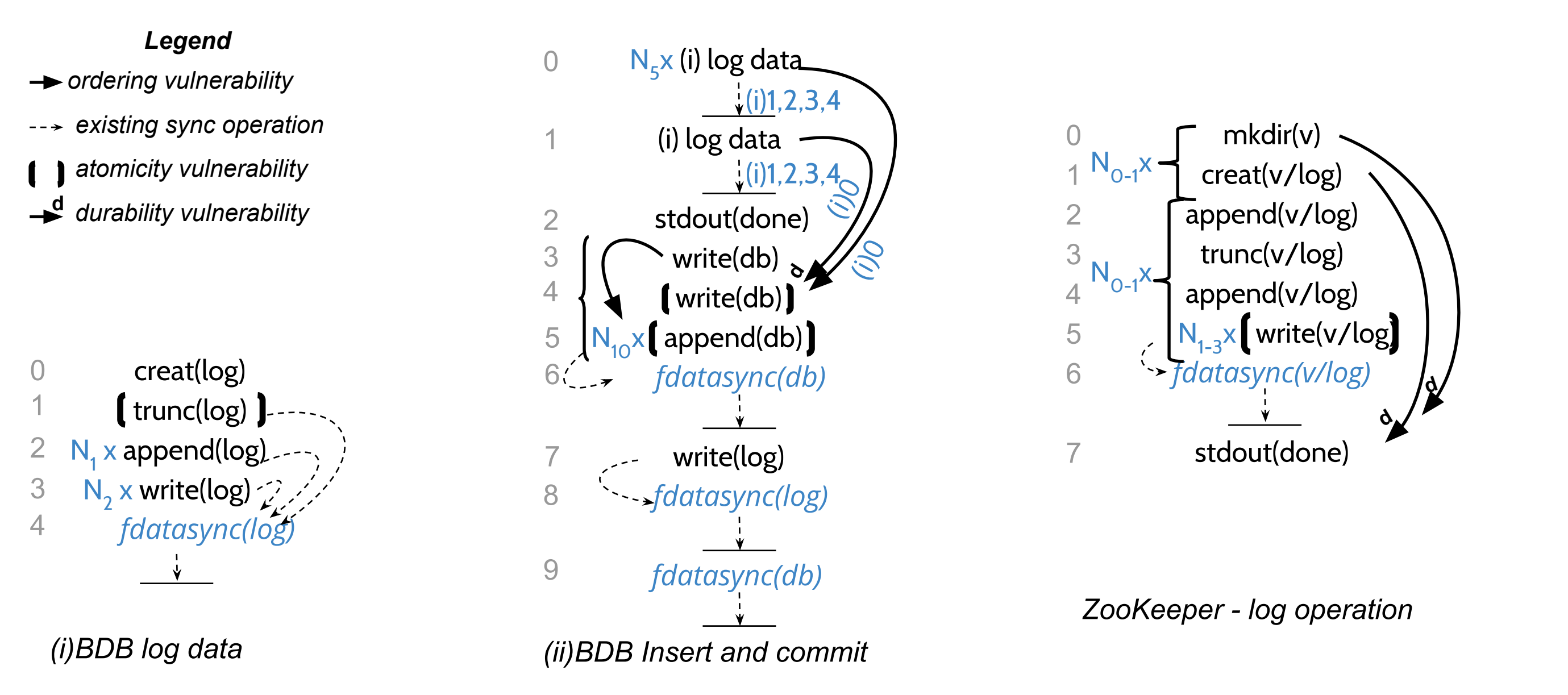
- User updates a file
- User wants update to be atomic
  - File should be fully in original state or updated state

Wrong Protocol

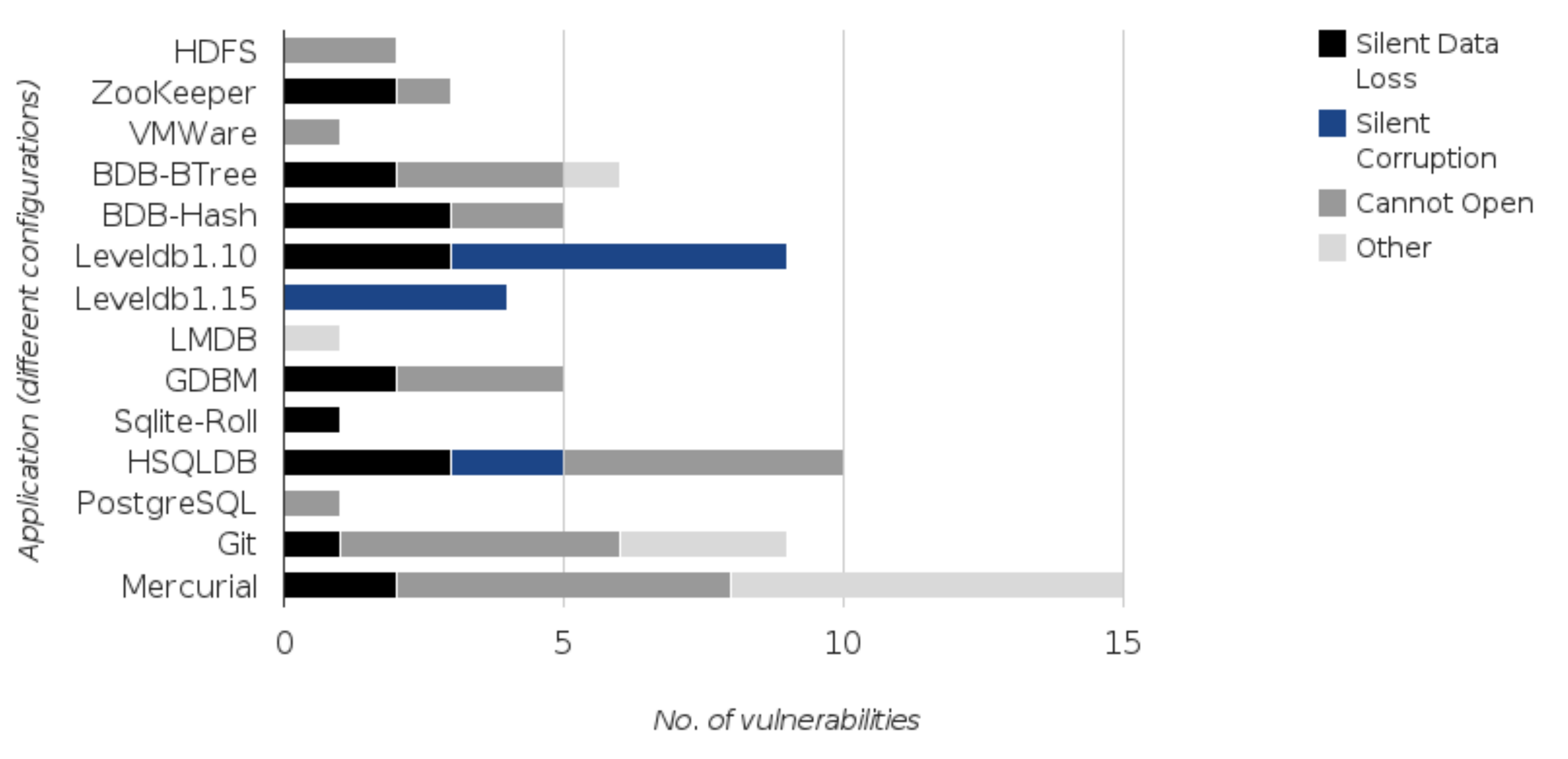
```
fd = creat("temp")
write(fd)
~~fsync(fd)~~
rename("temp", "grub.conf")
```

- Omitting fsync() might result in a zero-filled file
  - Because FS can re-order write() and rename()
- Wrong protocol is commonly used
- **Works under most common file systems**
  - Ext4, btrfs etc. explicitly ensure correctness
- Observation: FS implementation affects applications

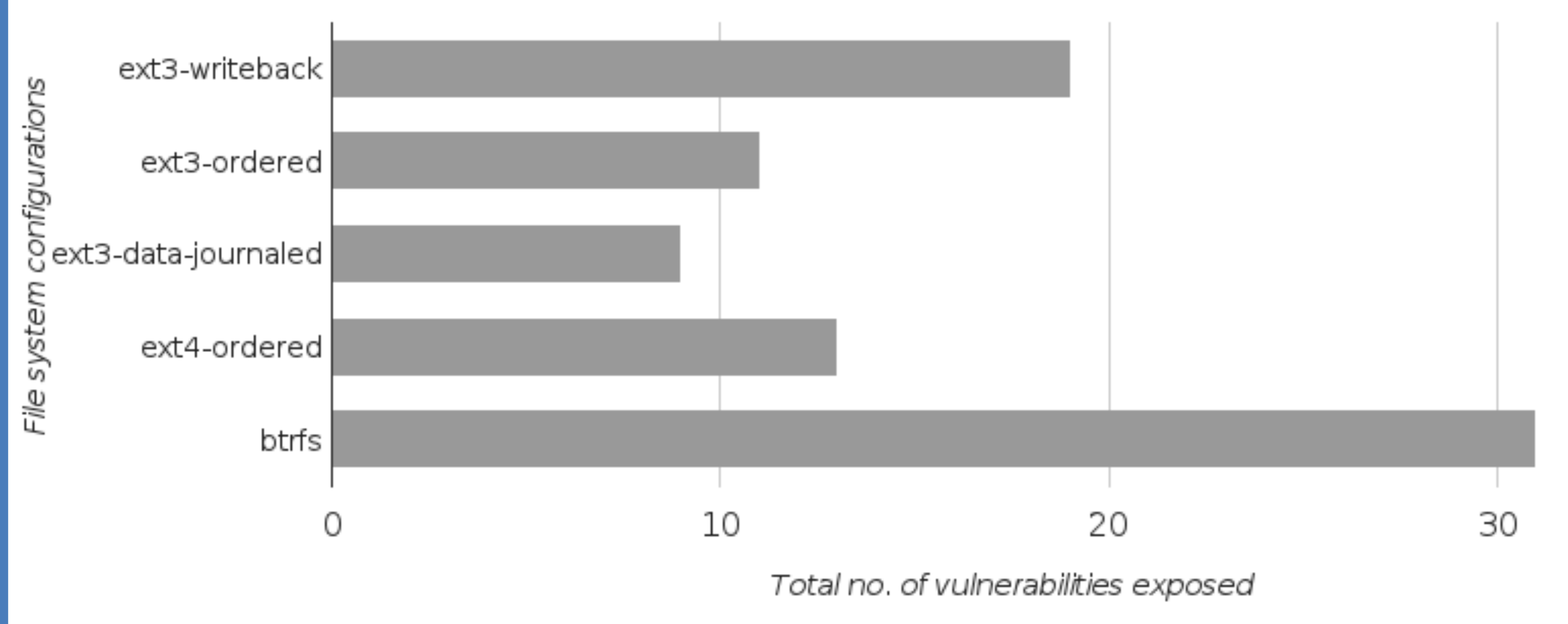
Discovered Vulnerabilities



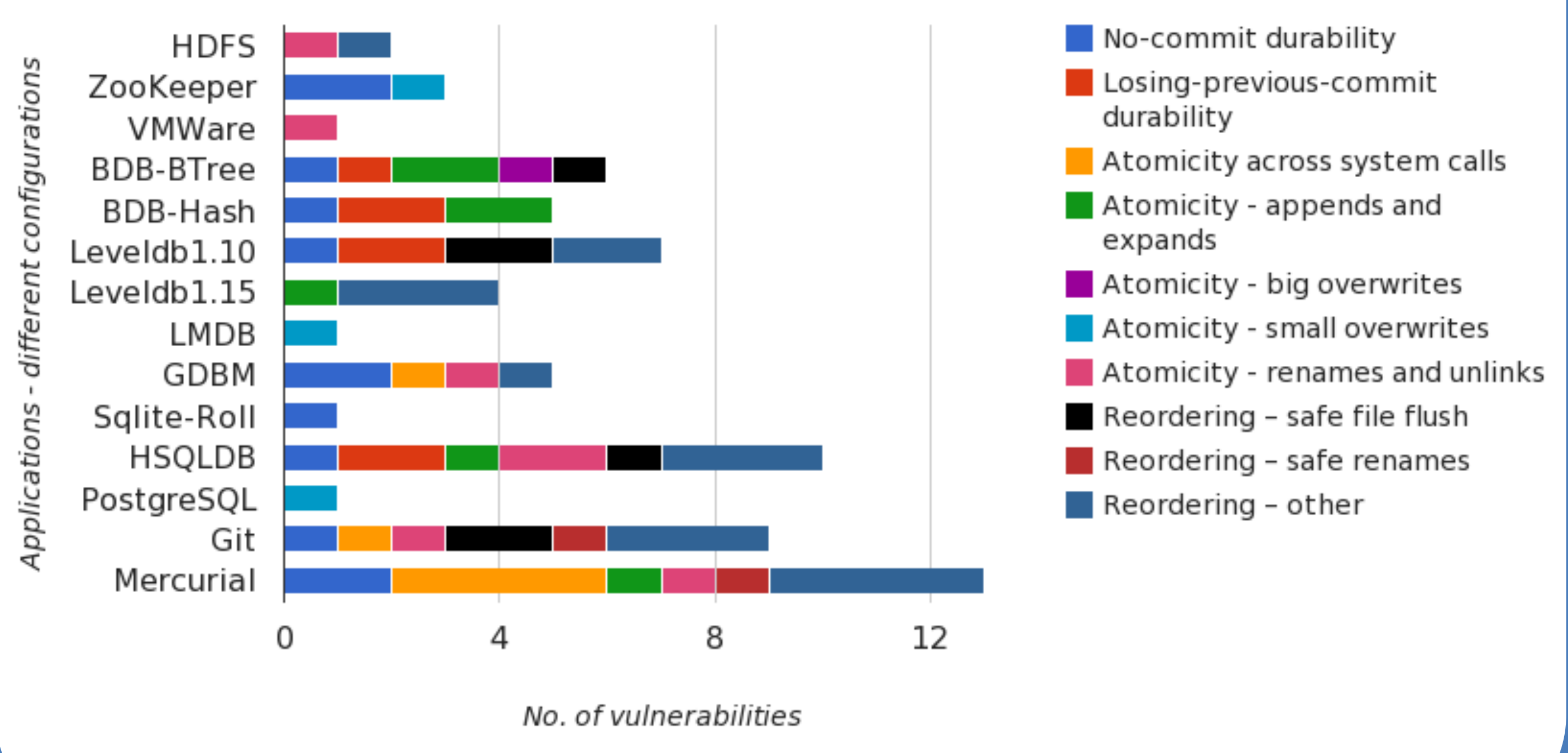
Total Unique Vulnerabilities: 65



Current FS Impact



Patterns



Variation of File System Implementation Details

Persistence Property of a File System (True / False):
Does a system call sequence only result in a given, desirable set of post-crash states?

Safe Rename

Atomic file rewrite is ensured even when omitting fsync() in the wrong protocol

System Call Sequence

```
fd = creat("temp")
write(fd)
rename("temp", "grub.conf")
```

Post-Crash states

grub.conf (Updated)

```
print "Hello"
kernel vmlinuz
initrd initrd.img
```

(or)

grub.conf (Original)

```

kernel vmlinuz
initrd initrd.img
```

~~*grub.conf* (Garbage)~~

```
#!@$%#!@$%#!@$%
#!@$%#!@$%#!@$%
#!@$%$%#!@$%
```

(or)

~~*grub.conf* (Zeroes)~~

```
0000000000000000
0000000000000000
0000000000000000
```

Safe Appends

When appending a file, appended part will never contain garbage

System Call Sequence

append(LogA, "1.00 Msg")

LogA (Original): 0.00 Started

(or)

LogA (Semi-updated): 0.00 Started / 1.00 M

(or)

LogA (Updated): 0.00 Started / 1.00 Msg

~~*LogA* (Garbage-appended): 0.00 Started / #!@$%#!~~

Ordered dir-ops

Directory operations (creat, unlink, rename …) get persisted in issued order

Safe new file

After fsync() on a new file, another fsync() on the parent directory is not needed

Ordered Appends

Append calls to files get persisted in issued order

System Call Sequence

```
append(LogA, "1.00 Msg")
append(LogB, "2.00 FAULT")
```

LogA: 0.00 Started	*LogA*: 0.00 Started / 1.00 Msg	*LogA*: 0.00 Started / 1.00 Msg	~~*LogA*: 0.00 Started~~
LogB: 0.00 Started	*LogB*: 0.00 Started	*LogB*: 0.00 Started / 2.00 FAULT	~~*LogB*: 0.00 Started / 2.00 FAULT~~

(or) (or)

File Systems	Safe rename	Ordered appends	Ordered dir-ops	Safe appends	Safe new file
ext3 – ordered	✓	✓	✓	✓	✓
ext3 – writeback	✓		✓		✓
ext4 – ordered	✓		✓	✓	✓
ext4 – original			✓	✓	✓
Btrfs	✓			✓	✓