Protecting a File System from Itself

Daniel Fryer (dfryer@cs.toronto.edu), Angela Demke Brown (demke@cs.toronto.edu), Ashvin Goel (ashvin@eecg.toronto.edu) – University of Toronto.

Motivation

Modern file systems are complex pieces of software, and complex software is highly likely to contain bugs. Current research into modern file systems indicate that bugs in the storage stack, including the file system, cause real data loss [1][2]. Checksums and redundancy, such as in ZFS, minimize the impact of bugs and physical defects below the file system but are unable to handle bugs in the file system itself [3]. Backups are prone to backing up the silently-corrupted data, providing no protection. On the other hand, the data stored on the disk adheres to a particular format. Its layout and the rules that describe its structure are usually fixed early on in the process of developing the file system and usually remain stable even as the code in the file system is modified. Tools like `fsck` exist to detect corruption of this structure and possibly repair it, or at least mitigate the damage. There are limitations to this approach – `fsck` must be run on an offline file system and can only detect damage after the fact. We aim to construct an online system to monitor disk I/O with the ability to ensure that writes don't corrupt the file system in the first place.

Implementation & Current Status

As a starting point for exploring our idea, we have designed a prototype for detecting known file system bugs, similar to vulnerability-specific predicates [4]. We chose a known bug in ext3 as a target and created an assertion in a layer below the file system that was able to detect the presence of the bug. The bug chosen was "Directory ctime not updated by rename", bug #10276 on the Linux kernel bug tracker at http://bugzilla.kernel.org. The bug made it possible to rename a file so that it was moved into a directory without updating the timestamp on the directory. This resulted in data loss when backup systems did not identify the directory as "changed" since, by timestamp, the directory appeared to be unmodified but the file was no longer present outside the directory. This problem is also undetectable by `fsck`, which is incapable of knowing when the relevant blocks were written to.

Our write-monitoring mechanism consists of two parts – a virtual device which snoops on all reads and writes to an underlying block device, and a modification of the ext3 journaling block device to invoke a check procedure before a transaction is committed to the journal. The function of the snoop layer is to collect file system metadata critical to checking the writes done in the journaling layer. When a transaction is about to commit, the check procedure scans it for blocks corresponding to the update of directory entries. If it finds a directory update, it verifies that the corresponding inode is also being updated with a reasonable creation & modification timestamp.

Future Work

A next step in the implementation of the project is to determine what kinds of data corruption bugs can be realistically caught, starting from a more thorough review of known bugs, and implement checks to catch their misbehavior. This will guide decisions about what metadata the snoop layer needs to gather. The ultimate goal is fully isolating the data on disk from faults in the file system. By creating a specification of a consistent file system structure and the online consistency guarantees implied by a file system, we hope to be able to determine whether or not a transaction will corrupt a file system. Because of performance considerations, it may not be reasonable to do some checks online; we believe that the ability to check consistency on-the-fly to be a highly desirable feature of future file systems.

References

[1] J. Yang, C. Sar, D. Engler. *eXplode: a Lightweight, General System for Finding Serious Storage System Errors.* OSDI '06

[2] L.N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau. *An Analysis of Data Corruption in the Storage Stack.* FAST '08.

[3] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau. *IRON File Systems.* SOSP '05.

[4] A. Joshi, S.T. King, G. W. Dunlap, P. M. Chen. *Detecting past and present intrusions through vulnerability-specific predicates.* SOSP '05