Managing Compliance Data: Addressing the Insider Threat Exemplified by Enron

Soumyadeb Mitra, Rick Snodgrass, Marianne Winslett
Department of Computer Science
University of Illinois at Urbana-Champaign
When something really bad happens, the government likes to quickly take action to reassure people that it will never happen again

- FDIC
- Sarbanes-Oxley Act
- $700B bailout
SOX had major repercussions for corporate IT

- Most people at the top got away with millions and many did no jail time ➔ top execs have to sign off on financial reports
- No paper trail available for prosecution ➔ retain routine business documents for (typically) 7 years, tamper-proof (*term-immutable*)
Compliance regulations have teeth: periodic audits, fines, jail terms

**SEC:** $1.65M each
- Deutsche Bank
- Goldman Sachs
- Morgan Stanley
- Solomon Smith Barney
- U.S. Bancorp

**SOX:**
- Rica Foods CEO $25K
- Deloitte $1M poor audit
The government likes to step in for non-corporate scandals as well.

- Video Privacy Protection Act of 1988
- Gramm-Leach-Bliley Act’s Financial Privacy Rule
- Health Insurance Portability and Accountability Act (HIPAA)
E-government records are also at risk for falsification.
WORM storage helps secure documents against insider tampering.

Commit File for Prespecified Retention Period

Append to File on certain volumes

Overwrite/Delete Unexpired File

Delete Expired File

Adversary cannot delete Alice’s file

Write Once, Read Many
WORM can be used for IM, email, spreadsheets, reports, and even indexes over them. But what about *structured* data?
The main “new” threat to tuples is undetected tampering with history.
The goal: a high-performance tamper-evident database that supports term-immutability.

Auditor must verify that final state is consistent with the initial state and sequence of transactions, even with crashes.
To support term-immutability, we’ll use a “transaction-time” database.

When tuple t is updated/deleted, create a timestamped new copy of it.

![Timestamps](image)

After 7 years: Shred!

**Legitimate update**: modifies/deletes the latest version

**Tampering**: modifies an old version, shreds unexpired tuple

**Shredding**: after expiration

No changes to existing DB applications
The database is *logically* append-only. Pages are modified *in place*.

Can be implemented atop an ordinary DBMS on ordinary disk.
Log-consistent DBMS: keep snapshot of DB and log of all new tuples on WORM.

Ordinary Disk

Ordinary Log

DB State

Snapshot

Final DB state

New Tuples (doesn’t delay commit)

(Trusted) auditor takes signed snapshot.
Space-efficient: delete after audit.
Auditor checks if every record in initial state and in the log is in the final state.

Then check snapshot signature, write & sign new snapshot. (Also validate integrity of pages, indexes, metadata.)
Tampering will make the compliance log and DB inconsistent.

Ordinary Disk

Snapshot DB state

WORM

DB State

Update t

T₁

T₂

T₃

Add “NEW_TUPLE <>”

Compliance Log

Final DB state

Integrity Check

Auditor
We can speed up audits by using existing B-trees during comparison.

1. Sort the compliance log on <relation name, primary key, timestamp>

2. Merge it with the old snapshot (already sorted) …

3. While comparing it to the new DB (already sorted)

Cost: $O(L \log L + D_S + D_f)$
We can make audits even faster with a commutative incremental cryptographic hash function

1. Compute $h(t)$, for all $t$
2. Add the $h(t)$ (mod large number)
3. Compare old and new sums
4. Store the new sum (and possibly the new (unsorted) snapshot)

Cost: $O(L + D_f)$
Compliance log records from aborted transactions will make the audit fail.

Memory Buffer Pool

Ordinary Disk

DB State

Snapshot DB state

WORM

DB State

Ordinary Log

Final DB state

NEW_TUPLE <t>

Compliance Log

NEW_TUPLE <t, transaction ID>

ABORT <transaction ID>

STAMP_TRANS <ID, time>

Auditor

Integrity Check
But queries between audits may read tampered values.
Record page hashes in the compliance log.

**Initial DB state**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Final DB state**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Compliance Log**

- NEW_TUPLE <t, page #>
- ABORT <transaction ID>
- STAMP_TRANS <ID, time>
- READ <page #, hash>
- SPLIT_PAGE <#, #, #, new contents>

**Integrity Check**

**Auditor**
The auditor can replay the log to compute the page hash.

Initial DB state

\[ p_1 \]

\[ T_1 \]

\[ p_2 \]

\[ T_2 \]

\[ T_3 \]

\[ T_n \]

Final DB state

Compliance Log

Recreate the hash

Integrity Check

Must hash EXACTLY what the reader saw:
Uncommitted tuples, missing timestamps, no already-aborted tuples.

Auditor
Tampering will cause the page hash to change.

Hash computed by the auditor from the compliance log won’t match the hash computed by the DBMS.
Replaying the log can be slow. Instead, use an incremental sequential hash function, assign each tuple an order # on its page.

Now only $O(L)$ to audit page reads. Additional complications to ensure that the auditor hashes EXACTLY what the reader saw.
Over time, the DB can get very big, making page integrity checks costly.

- Use time-split B-trees (Lomet & Salzberg) to separate out historical versions of tuples & their index entries
- Put historical tuples/index entries on WORM
- Only audit them one time on WORM
- Log changes to index pages as for data pages
The hard part: log-consistent DBs must handle crashes correctly

- Transaction committed but its entries are not in the compliance log
  - Flush the entries every *regret interval*

- Uncommitted transaction’s entries reach the compliance log
  - Entries must be *logically* removed from the log
  - The adversary should not be able to exploit this to delete records of committed transactions

- Recovery: put all new ABORT/STAMP_TRANS records on WORM before traditional recovery
Our implementation used Berkeley DB + transaction time layer + compliance plugin + time-split B-trees

- Not-quite-met goal: *don’t change BDB*
  - Log which transactions commit, abort
  - Clean up compliance log at beginning of recovery
  - Could implement these outside of kernel in future

- Logger taps into pread/pwrite
  - Compare new, old versions of page; differences go in compliance log
  - Hash page on pread \(\rightarrow\) trust the buffer cache

- TPC-C + tuple order #s, over NFS
Compliance logging and hash-on-read have very reasonable overhead.
Details, details, details in ICDE 2009 paper

- How to shred tuples (complicated but no fancy crypto)
- Non-quiescent audits
- Lazy/eager metadata changes
- Crash before committed transaction’s NEW_TUPLEs reach WORM
- Preventing attacks that exploit “quiet” DB times
- Duplicate NEW_TUPLE, UNDO entries due to crash recovery
- How to decide when to time split
- More experiments

…
In conclusion: we can provide term-immutability for RDBs at modest cost

- Keep signed DB snapshot, log of updates on WORM
  - TPC-C ~10% slower
  - 5-6.5 minute audit for 100K transactions
- Modest changes to DBMS kernel